Journal of Computers Vol. 35 No. 2, April 2024, pp. 215-230
doi: 10.53106/199115992024043502014

A Travel Salesman Problem Solving Algorithm Based on Feature Enhanced Attention Model

Xiaoxuan Ma[*], Chao Liu

School of Electrical and Information Engineering, Beijing University of Civil Engineering and Architecture, Beijing, China

maxiaoxuan@bucea.edu.cn, 2108550021016@stu.bucea.edu.cn

Received 27 March 2024; Revised 1 April 2024; Accepted 15 April 2024

Abstract. In order to address the challenges of low accuracy and weak generalization capabilities in solving the traveling salesman problem (TSP) using end-to-end deep reinforcement learning (DRL) algorithms, this paper introduced a novel solution. This solution consisted of a feature enhanced attention model (FEAM) and a rotation expanded inference method. The FEAM combined a feature filtering layer, a graph embedding layer, and Transformer architecture in the encoder to better capture the complex relationships between cities, obtain richer and more accurate node representations, and thereby improved the model's solving accuracy and generalization ability. The rotation expanded inference method generated new problem instances through coordinate rotation, enabling the model to consider path planning strategies from multiple perspectives, thus further enhancing solution accuracy. Numerous experiments on randomly generated benchmark datasets and public benchmark datasets show that the proposed end-to-end DRL algorithm performed better than other DRL algorithms in terms of solution quality and generalization ability.

Keywords: traveling salesman problem, deep reinforcement learning, feature enhanced attention model, rotation expanded inference method, combinatorial optimization problem

1 Introduction

The traveling salesman problem (TSP) is a well-known combinatorial optimization problem with wide applications in real-life scenarios such as transportation [1], scheduling [2], warehouse order picking [3], etc. The problem can be formulated as follows: given n cities and the distance d_{ij} between each pair of cities i and j, starting from an initial city, each city is visited exactly once, and the tour concludes by returning to the initial city. The objective is to find the shortest possible path.

With the rapid development of computing capabilities, deep learning techniques have been widely applied in various domains, including image processing [4], natural language processing, and more. Following the introduction of pointer networks [5], researchers have recognized the potential of neural networks in solving the TSP, leading to an increasing number of studies using both supervised learning [6, 7] and reinforcement learning [8-11] methods for the TSP. Supervised learning requires optimal solutions as labels, but constructing optimal solutions for TSP incurs high computational costs. In contrast, reinforcement learning does not rely on labels, offering a more computationally efficient approach. Therefore, recent research has shown a preference for utilizing deep reinforcement learning (DRL) algorithms to address the TSP.

DRL algorithms for solving TSP can be roughly categorized into two types based on the approach used to construct solutions: end-to-end DRL algorithms and search-based DRL algorithms. End-to-end DRL algorithms create paths from scratch, while search-based DRL algorithms typically start from existing solutions and continuously learn how to improve them. The solutions obtained from search-based methods usually outperform those from end-to-end methods. However, the performance of such methods heavily depends on the number of iterations or searches, which can increase time costs, making them unsuitable for time-sensitive tasks.

This paper focuses on end-to-end DRL algorithms because they are more suitable for real-world TSP applications, such as ride-sharing dispatch [12]. Such real-world applications often require high efficiency, necessitating the nearly real-time generation of solutions. However, most existing end-to-end DRL algorithms perform poorly in terms of solving performance and are not easily scalable to larger instances. The main reasons for this are: (1) Insufficient feature extraction of node characteristics in TSP instances, leading to the inability to select the opti-

* Corresponding Author

mal nodes subsequently. (2) The inference methods used by well-trained models in solving problem instances do not effectively leverage the model's performance, resulting in a failure to explore some better solutions.

To tackle the aforementioned issues, this paper proposes an end-to-end DRL method based on a Feature Enhanced Attention Model (FEAM). The method in this paper adopts the classical encoder-decoder architecture [13]. In the encoder part, a feature selection layer is designed to enable the model to learn crucial features, enhancing the model's expressiveness, robustness, and generalization capability. Additionally, a graph embedding layer is incorporated to facilitate the model in learning the neighbor relationships between nodes and the graph's topological structure, extracting and learning richer and more informative node representations from input features. Furthermore, this paper introduces a rotation-based expansion inference method to enlarge problem instances, thereby broadening the model's perspectives on the problem and consequently enhancing the quality of solutions.

To demonstrate the effectiveness of our proposed algorithm, we conducted experiments on two types of datasets: one consists of randomly generated TSP instances commonly used in previous DRL algorithm research, and the other comprises well-known TSP public datasets. The experimental results indicate that our algorithm achieves competitive results compared to existing deep reinforcement learning algorithms. Although the model was trained using randomly generated instances, it demonstrated excellent performance on various public datasets with different distributions. This indicates that our algorithm exhibits superior generalization capabilities.

The main contributions of this paper are as follows:

(1) We proposed FEAM. Expanded the Transformer style encoder decoder architecture by incorporating feature selection module and graph neural networks into the encoder to obtain more accurate and rich node representations, improving the model's expressive and generalization capabilities.

(2) We proposed a novel model inference method that enhances the problem-solving performance by flipping and rotating to expand the problem instance. This allows the model to approach problem-solving from multiple perspectives, thereby improving its overall performance.

(3) Extensive experiments demonstrate that our algorithm achieves competitive results in solving TSP instances of different scales. Additionally, our algorithm exhibits good generalization performance on public datasets with diverse distributions.

The structure of this paper is arranged as follows. Section 2 introduces related work on DRL algorithms. Section 3 presents the proposed method, providing specific details on problem formulation, model architecture, training methodology, and inference approach. In Section 4, we evaluate the proposed method using random TSP datasets and public TSP datasets, followed by a discussion of the experimental results. Finally, Section 5 summarizes the findings of this paper.

2 Related Work

This section introduces the relevant research on DRL algorithms for solving vehicle routing problems, which can be divided into two categories: improved local search DRL algorithms and end-to-end DRL algorithms.

2.1 Improved Local Search DRL Algorithms

The improved local search DRL algorithms primarily leverage DRL algorithms to automatically learn heuristic rules for local search, integrating DRL with local search algorithms.

Chen et al. [14] chose to train the strategy of local search using deep reinforcement learning algorithms, replacing manually designed heuristic rules with learned policies. They proposed a search model called NeuRewriter, which combines deep reinforcement learning. Initially, a random initial solution was constructed, and then the learned policy was used to select the way of local search. Through local search, the initial solution was optimized continuously, improving the quality of the solution.

Gao et al. [15] combined deep reinforcement learning with large neighborhood search algorithms, using deep reinforcement learning methods to learn the destruction and repair operators in large neighborhood search. The destruction operator removed nodes from the solution, and the repair operator added nodes to the solution. Both operators have multiple types for different forms of search. This method encoded the problem using graph attention neural networks and decoded the selection strategies of the destruction and repair operators.

Xin et al. [16] proposed NeuroLKH, an improved heuristic model using deep learning methods based on the LKH algorithm. The network model was a sparse graph network, trained with supervised and unsupervised learning to learn edge scores and node penalties. Based on the output of this network, an "edge candidate set" was created to guide the LKH search. Experimental results showed a significant improvement in performance.

2.2 End-to-end DRL Algorithms

Although the improved local search DRL algorithm can achieve good results, it requires a longer solution time. Many scholars have conducted research on more efficient end-to-end DRL algorithms.

Vinyals et al. [5] proposed the pointer network model for solving combinatorial optimization problems such as TSP in 2015, which initiated a series of studies on using deep neural networks to solve combinatorial optimization problems. Inspired by the Seq2Seq model in machine translation, this model used a deep neural network-based encoder to encode the input sequence (city coordinates) of TSP and then calculated the node selection probabilities through the decoder and attention mechanism. It progressively selected nodes in an autoregressive manner until a complete solution was obtained.

However, Vinyals et al. [5] trained the model in a supervised manner, requiring a large number of TSP instances and their corresponding optimal solutions. The difficulty in creating labels and constructing training samples consumed a significant amount of time. Moreover, this approach may result in the model's solution quality not surpassing that of the sample solutions. To address this issue, Bello et al. [17] chose to train the pointer network model using reinforcement learning algorithms and proposed the Neural Combinatorial Optimization (NCO) model. Specifically, each TSP instance was treated as a training sample, and the REINFORCE reinforcement learning algorithm was used for training. They introduced a Critic network to compute Baseline to reduce training variance. During the training process, the sequences output by the model can be improved continuously by adjusting the parameters of the value network trained by deep neural networks and introducing reward mechanisms. This method could solve larger-scale TSP instances, outperforming previous research [5] in solving TSP instances with 50 cities and approaching the optimal solution of Concorde in solving TSP instances with 100 cities.

After the introduction of the Transformer model [13], it quickly became a research hotspot in the field of natural language processing. The multi-head attention mechanism in the Transformer allowed for the extraction of deep features from different perspectives and dimensions, enabling nodes to propagate information through different channels. As a result, some scholars have begun to draw inspiration from the Transformer model for solving vehicle routing problems.

Deudon et al. [18] improved the pointer network model using the Transformer architecture. The encoder employed a structure similar to the Transformer model, utilizing the multi-head attention mechanism to obtain feature vectors for nodes. In the decoder, unlike the LSTM used in the pointer network model, the authors linearly mapped the selections of the last three steps to obtain a reference vector, thus reducing complexity. The model was trained using the classic REINFORCE reinforcement learning algorithm. Additionally, the authors employed a 2-opt local search strategy to optimize the output solutions of the network model, finding that this effectively improved the quality of solutions. The authors solved TSP problems with 20, 50, and 100 cities, yielding slightly better results than previous research [17].

Kool et al. [19] further improved on the basis of previous studies and proposed a new method that can solve multiple vehicle routing problems. While the encoder part remained unchanged, adopting the same structure as the Transformer model, the attention calculation method in the decoder part differed. Whereas previous research [18] utilized a calculation method similar to the classical pointer network model, the authors incorporated self-attention mechanisms, adding computational layers to enhance the model's performance. Additionally, improvements were made in the model training algorithm. Unlike the Critic baseline commonly used in previous studies, the authors utilized solutions obtained from a greedy policy as the baseline, referred to as the Rollout Baseline. The Rollout Baseline was generated by selecting the best-performing model from all policy models obtained during the previous training process. The Rollout Baseline was generated by selecting the best-performing model from all policy models obtained during the training process. Using a greedy policy for action selection, the target function value b(s) obtained by solving state s with this baseline policy was defined as the Rollout Baseline. This eliminated the need for separately adding a Critic network to calculate b(s). Experimental results demonstrated that this method outperforms all previous end-to-end models in solving TSP.

Additionally, some researchers have pursued solving the problem using graph neural networks. Nowak et al. [20] utilized a graph neural network (GNN) to estimate the probability of selecting each "edge," resulting in an adjacency matrix. This matrix was then transformed into the final path scheme using beam search. The model is trained using supervised learning, generating a large amount of training data with solvers like LKH3. The loss function is computed based on the true adjacency matrix of the labels and the model's output adjacency matrix. Khalil et al. [21] proposed a framework that combines reinforcement learning with graph embedding neural networks. They utilized the Structure2vec network to model the graph structure of the problem, computing the Q-values of the optional nodes and selecting them based on a greedy policy until a complete solution was obtained. The model is trained using the deep Q-learning algorithm.

To address the insufficient node feature extraction in existing end-to-end DRL algorithms, this paper improves the encoder based on previous studies [19]. Inspired by the channel attention mechanism [22], a feature selection layer is introduced into the encoder to enhance the learning ability of crucial features. Moreover, the graph neural network model, a powerful tool for handling graph data, has been applied to path optimization problems in prior research [23]. This paper further enhances the model's feature extraction capability by adding a graph embedding layer to the encoder. In terms of model inference methods, this paper draws inspiration from commonly used data augmentation techniques in computer vision. Building upon the 8x instance expansion proposed by Kwon et al. [24], the paper incorporates rotation operations to expand the instances to 16x, thus further enhancing the solution quality.

3 Method

3.1 Problem Formulation

This article primarily focuses on the classical TSP. Let $G(V, E)$ represent an undirected complete graph, where $V = \{v_i \mid 1 \le i \le N\}$ denotes all cities, N is the number of cities, and $E = \{e_{ij} \mid 1 \le i, j \le N\}$ represents the set of all edges. Let $C(i, j)$ denote the cost of moving from city v_i to v_j, representing the Euclidean distance between the two cities. The solution to the TSP is a feasible path that starts from one city, passes through all other cities exactly once, and eventually returns to the starting city. For a given path τ, its total cost $L(\tau)$ can be represented by Formula (1), where τ_i represents the i-th city on the path τ.

$$L(\tau) = \sum_{i=1}^{N-1} C(\tau_i, \tau_{i+1}) + C(\tau_N, \tau_1). \tag{1}$$

The construction method of the path τ is as follows: continuously select the next city from all cities yet to be visited until all cities have been visited, and then return to the starting city. This process can be viewed as a Markov Decision Process (MDP), where the decision at each step can be modeled using a neural network parameterized by θ. For a TSP instance s, the reward at each step is defined as the negative of the cost from the current city to the next city. Our goal is to maximize the cumulative reward, and the formula is as follows:

$$J(\theta|s) = \mathrm{E}_{\tau \sim p\ (\tau|s)} R(\tau). \tag{2}$$

$$R(\tau) = -L(\tau). \tag{3}$$

$$p_\theta(\tau|s) = \prod_{i=1}^{N} p_\theta(\tau_i | s, \tau_{1:i-1}). \tag{4}$$

Here, $\tau_{1:i-1}$ represents the path to the i-1 city.

The gradient of the objective function can be calculated using the policy gradient theorem. The specific formula is as follows:

$$\nabla_\theta J(\theta|s) = E_{p_\theta(\tau|s)}[\nabla_\theta \log p_\theta(\tau|s) R(\tau)]. \tag{5}$$

In practice, Monte Carlo method is commonly used to estimate the expectation value. This involves sampling trajectories and calculating the expected value of gradients to approximate the true policy gradient. The formula is as follows:

$$\nabla_\theta J(\theta|s) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log p_\theta(\tau[i]|s)R(\tau[i]). \quad \textbf{(6)}$$

Here, N is the number of sampled trajectories, and $\tau[i]$ represents the i-th sampled trajectory.

3.2 FEAM Architecture

The structure of the proposed FEAM in this paper is depicted in Fig. 1.

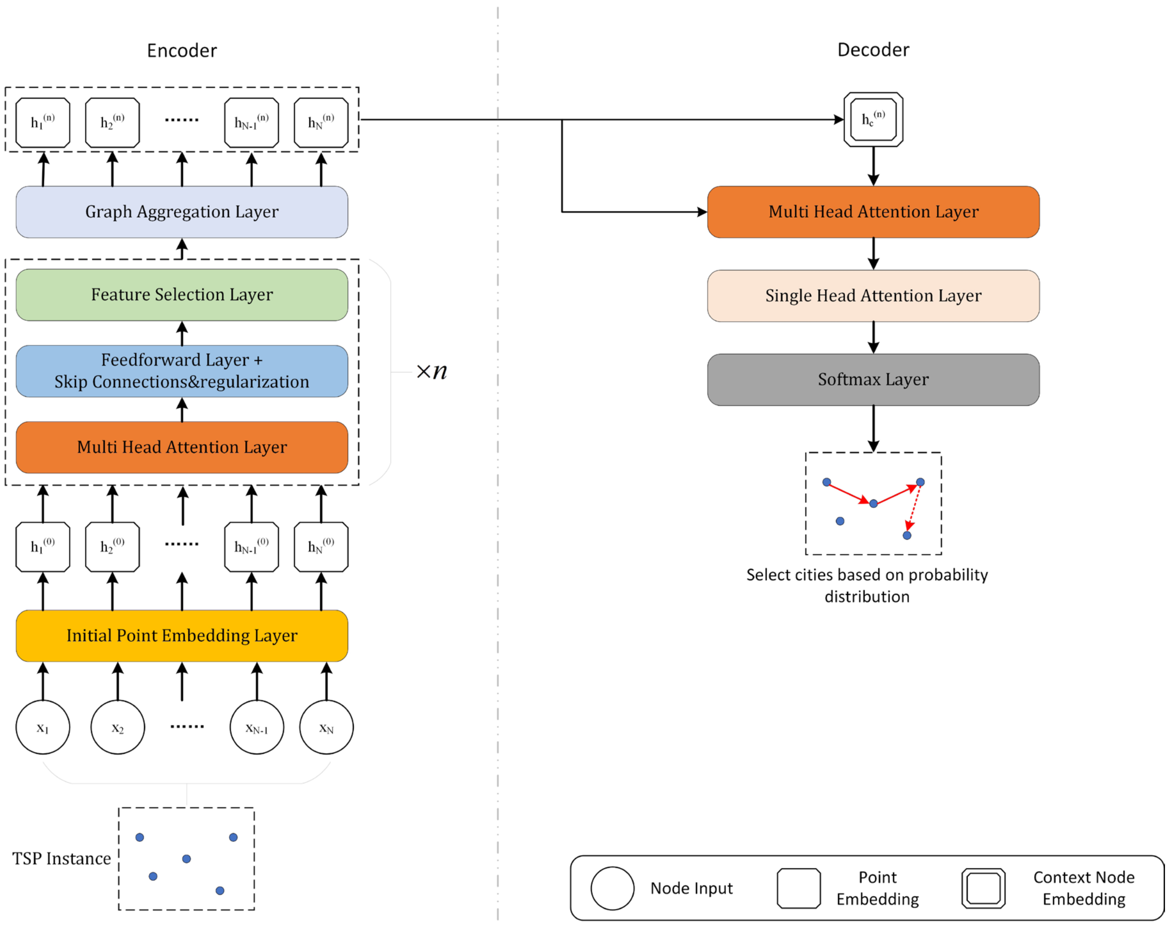
Encoder
Decoder
$h_1^{(n)}$ $h_2^{(n)}$ ...... $h_{N-1}^{(n)}$ $h_N^{(n)}$
Graph Aggregation Layer
Feature Selection Layer
Feedforward Layer + Skip Connections&regularization
Multi Head Attention Layer
×n
$h_1^{(0)}$ $h_2^{(0)}$ ...... $h_{N-1}^{(0)}$ $h_N^{(0)}$
Initial Point Embedding Layer
x_1 x_2 ...... x_{N-1} x_N
TSP Instance
$h_c^{(n)}$
Multi Head Attention Layer
Single Head Attention Layer
Softmax Layer
Select cities based on probability distribution
Node Input
Point Embedding
Context Node Embedding

Fig. 1. FEAM structure

FEAM utilizes multiple attention layers consisting of multi-head attention layers, feedforward layers, and feature selection layers to encode input nodes. Subsequently, a graph aggregation layer is employed to aggregate node information, obtaining embedding for each node. Then, based on queries formed by context embedding, a single-head attention calculates the selection probability for each node. Using the probability distribution output by the decoder, a node with the highest probability is selected at each step to be added to the solution. Nodes that have already been selected are blocked, and their probabilities are set to 0 to prevent duplicate selections. Finally, the model is trained using an improved reinforce algorithm, which incorporates a variance normalization mechanism. This section will provide a detailed description of the key components of FEAM.

Encoder. Due to the relatively weak feature extraction capability of the encoder in previous studies [19], we have improved the encoder by introducing feature selection modules and graph aggregation layers to obtain richer node information. The structure of the encoder is illustrated in the left half of Fig. 1.

The coordinates of each point in the TSP can be represented as (x, y), indicating that the initial input point feature x_i is two-dimensional. The initial point embedding layer linearly projects the two-dimensional input feature x_i onto node embedding $h_i^{(0)}$ with dimension d_h. Subsequently, n attention layers are used to update the point embedding. Each attention layer consists of three components: multi-head attention (MHA), feed-forward (FF), and feature selection layer. Let $h_i^{(l)}$ represent the point embedding for the i-th node after the l-th layer in the attention layer, and $\{h_1^{(l-1)}, h_2^{(l-1)}, ..., h_N^{(l-1)}\}$ be the output of the (l-1)-th layer, serving as the input for the l-th layer. Multi-head attention (MHA) has demonstrated excellent performance in previous work [25] and is used here to extract information from different nodes. The number of heads is denoted as M.

The MHA can be expressed as:

$$q_{im}^{(l)} = W_m^Q h_i^{(l-1)}, k_{im}^{(l)} = W_m^K h_i^{(l-1)}, v_{im}^{(l)} = W_m^V h_i^{(l-1)}. \tag{7}$$

$$u_{ijm}^{(l)} = \frac{(q_{im}^{(l)})^T k_{jm}^{(l)}}{\sqrt{d_k}}. \tag{8}$$

$$a_{ijm}^{(l)} = \frac{e^{u_{ijm}^{(l)}}}{\sum_{j'=1}^{N} e^{u_{ij'm}^{(l)}}}. \tag{9}$$

$$h'^{(l)}_{im} = \sum_{j=1}^{N} a_{ijm}^{(l)} v_{jm}^{(l)}. \tag{10}$$

$$MHA_i^{(l)}(h_1^{(l-1)}, h_2^{(l-1)}, ..., h_N^{(l-1)}) = \sum_{m=1}^{M} W_m^O h'^{(l)}_{im}. \tag{11}$$

Here, $m \in \{1, 2, ..., M\}$, $i, j \in \{1, 2, ..., N\}$. Formula (7) represents the calculation method for the query vector $q_{im}^{(l)}$, key vector $k_{im}^{(l)}$, and value vector $v_{im}^{(l)}$ of node i. The dimensions of the query and key vectors are d_k, and the dimension of the value vector is d_v. W_m^Q, W_m^k, and W_m^v are trainable parameters. Formula (8) represents the calculation of attention scores, and it includes scaling [13] to enhance the stability of attention computation. Formula (9) represents the use of the softmax function to calculate attention weights. Formula (10) represents the calculation of the node vector for node i in a single attention head. Formula (11) represents the calculation of the node vector for node i when using multiple attention heads, where W_m^O is a trainable parameter.

After the MHA, the node vectors computed through multi-head attention enter the feedforward layer. Following this, skip connections [26] and regularization operations are applied, and the calculation formula is:

$$\hat{h}_i^{(l)} = \text{Norm}(h_i^{(l-1)} + MHA_i^{(l)}(h_1^{(l-1)}, h_2^{(l-1)}, ..., h_N^{(l-1)})). \tag{12}$$

$$FF(\hat{h}_i^{(l)}) = W_1^F \text{ReLu}(W_0^F h_i^{(l)} + b_0^F) + b_1^F. \tag{13}$$

$$h'^{(l)}_i = \text{Norm}(\hat{h}_i^{(l)} + FF(h_i^{(l)})). \tag{14}$$

Here, $\mathrm{Norm}(\cdot)$ represents regularization, and W_1^F , W_0^F , b_0^F , and b_1^F are trainable parameters.

Unlike previous work, we do not directly output node embedding after passing through the feedforward layer. Instead, we pass the node embedding into the feature selection layer. The role of the feature selection layer is to enhance the representation capability of input features by dynamically adjusting the weights between different features. This allows the network to better focus on important features. It helps the model better understand the importance of each node, leading to more accurate path selection. The feature selection layer is illustrated in Fig. 2.

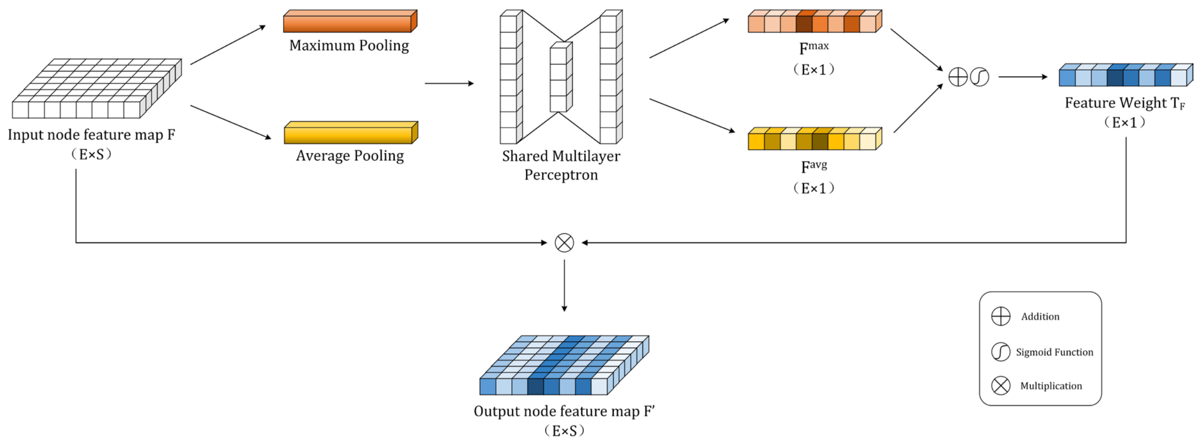

Fig. 2. Illustration of the feature selection layer

Given the node feature map $\mathrm{F} \in \mathbb{R}^{E \times S}$ as input, where E is the node embedding and S is the number of nodes. First, aggregate spatial information from the feature map by applying average pooling and max pooling operations. Then, pass the result into a shared perceptron, outputting two spatial descriptors, $\mathrm{F}^{\max} \in \mathbb{R}^{E \times 1}$ and $\mathrm{F}^{\mathrm{avg}} \in \mathbb{R}^{E \times 1}$. Next, add these two descriptors, pass them through a sigmoid function, and generate feature weights $\mathrm{T_F} \in \mathbb{R}^{E \times 1}$. Finally, multiply the input node feature map by the feature weights to obtain the weighted node feature map. From the weighted node feature map, point embedding $h''^{(l)}_i$ for each point can be extracted.

After the entire attention layer processing, to better capture the relationships between nodes and the graph's topological structure for richer node embedding, we utilize a graph aggregation layer for node feature propagation. The graph aggregation layer employs graph convolution, which is a convenient and effective approach. Following the graph aggregation layer, the encoder outputs the final node embedding, denoted as $h_i^{(n)}$.

Decoder. The decoder is a self-regressive process that sequentially selects visiting nodes for the TSP instance, ultimately forming a feasible route.

The contextual node embedding are used to represent the current state and interact with candidate nodes as queries. As information about the starting and last visited nodes is crucial for selecting the next node, these details are included in the contextual node embedding. This is akin to previous research [19]. The contextual node embedding $h_c^{(n)}$, along with embedding of unselected nodes, are input into the MHA. This calculates a new, more enriched and comprehensive contextual node embedding $h_c^{(n+1)}$. Here, the MHA is the same as described in the encoder section. Finally, the single-head attention layer and softmax layer are utilized to obtain the final probability distribution, expressed by the following formula:

$$q_c = W^Q h_c^{(n+1)}, k_i = W^K h_i^{(n)}. \tag{15}$$

$$u_j = \begin{cases} C \cdot \tanh(\frac{q_c^T k_j}{\sqrt{d_k}}), & \text{node } j \text{ has not been visited} \\ -\infty, & \text{other} \end{cases}. \tag{16}$$

$$p_\theta(\tau_i | s, \tau_{1:i-1}) = \frac{e^{u_i}}{\sum_{j=1}^{n} e^{u_j}}. \tag{17}$$

Where, Formula (16) utilizes the tanh function and masks all previously visited nodes [17]. This ensures that the model only selects nodes that have not been visited. C is a coefficient controlling the range of values. Formula (17) uses the softmax function to calculate the selection probability of the node.

3.3 Model Training

The REINFORCE algorithm [27] is a classic reinforcement learning algorithm, and many existing models use this algorithm to train policy parameters, demonstrating good performance. Therefore, we use this algorithm to train the FEAM.

Inspired by previous research [24], we adopt a “multi-start” approach for training. “Multi-start” means that during the decoding process, each point in the TSP instance is considered as a starting point. In other words, for a TSP instance s_i with N nodes, we can sample N feasible paths, denoted as $\tau_i = \{\tau_i^1, \tau_i^2, ..., \tau_i^N\}$.

These paths can be trained using the REINFORCE algorithm, but due to significant differences in rewards between different instances, applying it directly may lead to convergence challenges. Therefore, we introduce a variance-normalized mechanism for stability in training. The gradient update can be expressed as:

$$\nabla_\theta J(\theta) \approx \frac{1}{BN} \sum_{i=1}^{B} \sum_{j=1}^{N} (\frac{R(\tau_i^j) - a(\tau_i)}{v(\tau_i)}) \nabla_\theta \log p_\theta(\tau_i^j | s_i). \tag{18}$$

Here, B represents the batch size, $a(\tau_i)$ and $v(\tau_i)$ respectively denote the mean and variance of N different starting point paths for the instance s_i.

The model training process is outlined in Table 1.

Table 1. FEAM training algorithm

Algorithm 1. FEAM training algorithm

Input: training set S, number of starting nodes N for each instance, training steps T, iteration count E, batch size B.

Output: policy network parameters θ.

```
1: Initialize policy network parameters $\theta$
2: for $iter$ = 1 to $E$ do
3:   for $step$ = 1 to $T$ do
4:     Retrieve instance $s_i$ from $S$, $\forall i \in \{1,2,...,B\}$
5:     Select the starting node $\{a_i^1, a_i^2, ..., a_i^N\}$ for $s_i$, $\forall i \in \{1,2,...,B\}$
6:     Compute solution $\tau_i^j$ using FEAM, $\forall i \in \{1,2,...,B\}$, $\forall j \in \{1,2,...,N\}$
7:     Compute solution $\nabla_\theta J(\theta)$ using Formula (16)
8:     Update parameters $\theta$, $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
9:   end for
10: end for
```

3.4 Inference Method

In existing DRL methods for solving the TSP, various reasoning techniques are often combined to improve the quality of solutions, such as greedy search, beam search, sampling search, and more. Literature [24] introduces a more effective 8-fold augmentation inference method by applying symmetry transformations to the coordinates of points in a single TSP instance, resulting in 8 distinct TSP instances. The proposed specific transformation method is shown in Table 2, considering that the coordinates of points in the training and testing datasets fall within the range of [0, 1].

Table 2. Transformation of point coordinates

Before transformation	After transformation	
(x, y)	(x, y)	(y, x)
	$(1\text{-}x, y)$	$(1\text{-}y, x)$
	$(x, 1\text{-}y)$	$(y, 1\text{-}x)$
	$(1\text{-}x, 1\text{-}y)$	$(1\text{-}y, 1\text{-}x)$

Although the coordinates of points in these 8 instances are different, the relative positions of each point remain unchanged. Therefore, the optimal solution is consistent across these instances. The model is used to solve these 8 instances, and the best solution is selected for output. These transformations are similar to data augmentation methods in the field of computer vision, with commonly used data augmentation techniques such as flipping and rotating. Inspired by this, we propose the rotation augmentation inference method, which further enlarges the TSP instances to 16 times by introducing rotation operations. This allows the model to consider the problem from more perspectives, aiming to enhance the quality of solutions.

We rotate the nodes in the TSP instance counterclockwise by an angle of γ around the origin, forming a new TSP instance, as illustrated in Fig. 3.

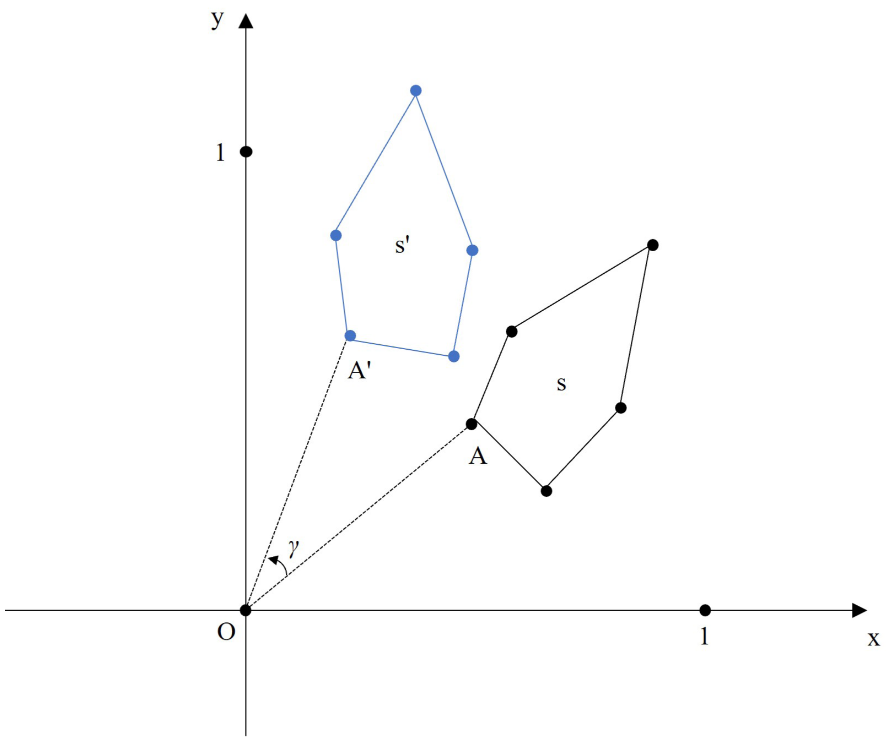

Fig. 3. Obtain a new instance s′ through rotation

The figure illustrates the rotation method with an example of 5 nodes. After counterclockwise rotation by an angle of γ, the 5 nodes in instance *s* transform into a new instance *s'*. Suppose point A in instance s has coordinates (*x*, *y*), the coordinate calculation formula for the rotated point A' is as follows:

$$\begin{cases} x' = x\cos(\gamma) - y\sin(\gamma) \\ y' = x\sin(\gamma) + y\cos(\gamma) \end{cases}. \tag{19}$$

Although the positions of the points have changed, the relative positions between the points have not changed. The optimal solution for instances *s* and *s'* is identical. Additionally, while some points may go beyond the range of the training data after rotation, we believe that the impact on the model's performance is limited, as the model inherently possesses a certain degree of generalization capability. The rotation augmentation inference method allows us to increase the augmentation from the original 8 times to 16 times. The inference steps of the model are outlined in Table 3. We validated the effectiveness of this method through ablation experiments and identified the optimal value for γ. Details can be found in Section 4.4.

Table 3. FEAM inference algorithm

```
Algorithm 2. FEAM inference algorithm
Input: test instance $s$, the number of starting nodes $N$, and the augmentation factor $K$.
Output: optimal path.
1: Extend one instance $s$ into $K$ instances, denoted as $\{s_1, s_2, ..., s_K\}$
2: Select the starting node $\{a_k^1, a_k^2, ..., a_k^N\}$ for instance $s_k$, $\forall k \in \{1,2,...,K\}$
3: Calculate the solution $\tau_k^i$ using the trained FEAM, $\forall i \in \{1,2,...,N\}$, $\forall k \in \{1,2,...,K\}$
4: Find a pair $(k, i)$ that maximizes the reward value $R(\tau_k^i)$, denoted as $k_{max}$ and $i_{max}$
5: Return $\tau_{k_{\max}}^{i_{\max}}$
```

4 Experiment

To evaluate the performance of the proposed method in this paper, we compared it with other DRL methods. Firstly, we implemented the algorithm based on PyTorch [28], and trained and tested the model on a single Tesla V100 GPU under the Windows 10 operating system. Then, we trained the model on randomly generated instances, and in testing, in addition to a randomly generated test set, we also used public datasets to test the generalization of the model.

4.1 Datasets

This paper utilized two datasets, namely the random TSP dataset and the public TSPLIB dataset.

(1) Random TSP dataset

This type of dataset is widely used in existing DRL research. It involves uniformly selecting a certain number of nodes from a $[0,1]\times[0,1]$ unit square to generate random TSP instances. This paper uses four datasets with different sizes, namely 20, 50, 100, and 200 nodes, denoted as TSP20, TSP50, TSP100, and TSP200, respectively. Following previous research, for TSP instances with 100 nodes or fewer, 10,000 instances are generated as the test set, and for TSP instances with 200 nodes, 128 instances are generated as the test set.

(2) TSPLIB dataset

This is a well-known TSP instance library, and the instances in it come from the real world. Consistent with literature [29], we selected 33 instances from it as the test set to evaluate the performance of the proposed method on real-world problems.

4.2 Hyperparameter Settings

In the experiments, only the random TSP dataset was used for model training. In each training batch, 100,000 instances were generated as the training set. For training the TSP200 model, the batch size was set to 32, while for other smaller instances, the batch size was set to 64. The Adam optimizer was used, with the learning rate η = 1×10^{-4} and weight decay $\omega = 1\times10^{-6}$. In the encoder, the dimension of node embedding $d_h = 128$, the number of layers in the attention layer is 6, the number of attention heads in the MHA is 8, and both d_k and d_v are set to 16.

4.3 Comparison of Experimental Results

Random TSP Dataset Experiment. To validate the effectiveness of our proposed method, we trained models for different sizes of TSP and tested them using corresponding random TSP test sets.

Our baseline algorithms fall into three categories: traditional algorithms, end-to-end DRL algorithms, and search-based DRL algorithms. Traditional algorithms include the exact algorithm solver Concorde and the heuristic algorithm LKH-3 [30]. End-to-end DRL algorithms include AM [19], POMO [24], AM-LCP [29], and CNN-Transformer [31]. Search-based DRL algorithms include DRL-2opt [32] and Att-GCRN+MCTS [33]. As our inference method is an improvement on top of the 8x expansion of POMO, for a fair comparison, we reproduced POMO and used our proposed inference method (which enhances the performance of POMO). Additionally, we presented results from traditional algorithms and AM as reported in the literature [33], while the remaining results are from the original papers. The experimental results on the random TSP dataset are shown in Table 4, where "Len" denotes the average length of all test instances, "Gap" represents the gap with respect to the optimal solution, and "T" indicates the time to solve the entire test set. "-" denotes cases where the corresponding results were not provided in the original literature.

Table 4. Comparison of results from different methods on random TSP test dataset

Algorithm	TSP20			TSP50			TSP100			TSP200		
	Len	Gap	T	Len	Gap	T	Len	Gap	T	Len	Gap	T
Concorde	3.83	0.00%	2.31m	5.69	0.00%	13.68m	7.76	0.00%	1.04h	10.72	0.00%	3.44m
LKH-3	3.83	0.00%	20.96m	5.69	0.00%	26.65m	7.76	0.00%	49.96m	10.72	0.00%	2.01m
DRL-2opt	3.84	0.26%	15.00m	5.70	0.18%	29.00m	7.87	1.39%	41.00m	-	-	-
Att-GCRN+MCTS	**3.83**	0.00%	1.64m	**5.69**	0.01%	7.92m	**7.76**	0.03%	14.56m	10.81	0.88%	2.49m
AM	3.83	0.05%	16.47m	5.72	0.49%	22.85m	7.97	2.73%	1.23h	11.44	6.82%	4.49m
POMO	3.83	0.14%	7.92s	5.69	0.02%	33.24s	7.77	0.16%	**2.18m**	11.03	2.91%	11.70s
AM-LCP	3.84	0.26%	30.00m	5.70	0.18%	6.89h	7.81	0.64%	11.94h	-	-	-
CNN-Transformer	-	-	-	5.70	0.10%	-	7.85	1.11%	1.43h	-	-	-
FEAM (ours)	**3.83**	0.00%	**6.65s**	**5.69**	0.01%	**29.89s**	7.77	0.12%	2.20m	**10.80**	0.73%	**11.51s**

From Table 4, it is evident that our approach has yielded competitive results. Comparing with traditional algorithms, the gaps between FEAM and traditional algorithms are 0.00%, 0.01%, 0.12%, and 0.73% for TSP20, TSP50, TSP100, and TSP200, respectively. While there is some difference in results, FEAM requires significantly less time in terms of runtime compared to traditional algorithms.

In comparison with DRL algorithms, FEAM achieved optimal results in the TSP20, TSP50, and TSP200 tests. In TSP20 and TSP50 tests, Att-GCRN+MCTS also demonstrated favorable results, but the required runtime was noticeably higher than FEAM. In larger-scale tests like TSP100 and TSP200, FEAM and Att-GCRN+MCTS had comparable results, but FEAM consumed less time. Compared to other DRL algorithms, FEAM strikes a better balance between efficiency and optimality.

TSPLIB Dataset Experiment. To evaluate the generalization capability of our method, we selected 33 instances from TSPLIB, where the node distributions differ from the training set. Similar to previous research [29], all algorithms were tested using models trained on a fixed scale (100 nodes). In contrast to experiments with the random TSP dataset, we did not compare with Att-GCRN+MCTS and CNN-Transformer here because there are currently no identified methods using these two algorithms to solve non-random TSP instances. The final experimental results are shown in Table 5.

Table 5. Comparison of results from different methods on TSPLIB

Instance	AM			DRL-2opt			POMO			AM+LCP			FEAM (ours)		
	Len	Gap	T	Len	Gap	T	Len	Gap	T	Len	Gap	T	Len	Gap	T
eil51	435	2.11%	13s	**427**	0.23%	460s	432	1.41%	0.03s	429	0.73%	13s	429	0.73%	0.03s
berlin52	8663	14.86%	14s	7974	5.73%	460s	7572	0.40%	0.05s	7550	0.10%	13s	**7544**	0.02%	0.04s
st70	690	2.18%	23s	680	0.74%	540s	**677**	0.30%	0.06s	680	0.74%	13s	**677**	0.30%	0.05s
eil76	555	3.18%	27s	552	2.60%	540s	**544**	1.12%	0.07s	547	1.64%	18s	**544**	1.12%	0.06s
pr76	110956	2.59%	27s	111085	2.60%	540s	**108159**	0.00%	0.06s	108633	0.44%	18s	**108159**	0.00%	0.05s
rat99	1309	8.09%	44s	1388	14.62%	680s	1252	3.36%	0.07s	1292	6.67%	24s	**1237**	2.15%	0.06s
rd100	8137	2.87%	46s	7944	0.43%	680s	**7910**	0.00%	0.07s	7920	0.13%	26s	**7910**	0.00%	0.07s
kroA100	23227	9.14%	46s	23751	11.60%	680s	21539	1.21%	0.09s	21910	2.95%	26s	**21456**	0.82%	0.07s
KroB100	23227	8.23%	46s	23790	7.45%	680s	**22254**	0.51%	0.08s	22476	1.51%	26s	22374	1.05%	0.07s
KroC100	21868	5.40%	46s	22672	9.27%	680s	20829	0.38%	0.08s	21337	2.84%	26s	**20764**	0.07%	0.07s
KroD100	22984	7.94%	46s	23334	9.58%	680s	21998	3.31%	0.08s	21714	1.97%	26s	**21669**	1.76%	0.07s
KroE100	22686	2.80%	46s	23253	5.37%	680s	22296	1.03%	0.07s	22488	1.90%	26s	**22294**	1.02%	0.07s
eil101	654	4.03%	46s	**635**	0.95%	680s	651	3.50%	0.07s	645	2.59%	26s	641	1.91%	0.07s
lin105	16516	14.87%	49s	16156	12.36%	680s	**14505**	0.88%	0.08s	14934	3.86%	26s	14558	1.24%	0.08s
pr124	63931	8.30%	68s	59516	0.82%	700s	**59247**	0.37%	0.10s	61294	3.84%	37s	59350	0.54%	0.09s
bier127	125256	5.90%	72s	**121122**	2.40%	720s	125472	6.07%	0.11s	128832	8.92%	37s	122280	3.38%	0.09s
ch130	6279	2.76%	77s	6175	1.06%	790s	6123	0.21%	0.11s	6145	0.57%	38s	**6117**	0.11%	0.10s
pr136	101927	5.33%	84s	98453	1.74%	820s	**97774**	1.04%	0.11s	98285	1.56%	38s	97823	1.09%	0.10s
pr144	63778	8.95%	93s	61207	4.56%	720s	58826	0.49%	0.11s	60571	3.47%	43s	**58802**	0.45%	0.11s
KroA150	28658	8.05%	102s	30078	13.40%	900s	27254	2.75%	0.12s	27501	3.68%	44s	**26959**	1.64%	0.11s
KroB150	27565	5.49%	102s	28169	7.80%	900s	26814	2.62%	0.12s	26962	3.18%	44s	**26746**	2.36%	0.11s
pr152	79442	7.82%	101s	75301	2.20%	720s	74447	0.79%	0.13s	75539	2.52%	44s	**74040**	0.49%	0.12s
u159	50656	20.38%	111s	42716	1.51%	840s	**42543**	1.10%	0.13s	46640	10.84%	45s	42920	1.20%	0.13s
rat195	**2518**	8.14%	168s	2955	27.21%	1080s	2552	9.86%	0.19s	2574	10.81%	57s	2545	9.56%	0.18s
KroA200	33313	13.43%	173s	32522	10.74%	1120s	**29971**	2.05%	0.20s	31172	6.14%	86s	30209	2.86%	0.19s
ts225	138000	8.97%	223s	127731	0.86%	1110s	127827	0.93%	0.22s	134827	6.46%	113s	**127824**	0.93%	0.21s
tsp225	4837	23.42%	224s	4354	11.10%	1160s	4145	5.77%	0.22s	4487	14.50%	113s	**4138**	5.59%	0.21s
pr226	90390	12.47%	228s	91560	13.92%	940s	**82654**	2.84%	0.24s	85262	6.09%	113s	82733	2.94%	0.22s
gil262	2588	8.81%	306s	2490	4.71%	1380s	2533	6.52%	0.33s	2508	5.49%	134s	**2453**	3.15%	0.30s
lin318	47288	12.51%	397s	46065	9.60%	1470s	44747	6.47%	0.45s	46540	10.72%	158s	**44259**	5.31%	0.43s
rd400	17053	11.59%	458s	**16159**	5.75%	1870s	17085	11.81%	0.72s	16519	8.10%	209s	16402	7.34%	0.70s
pr439	160594	49.78%	744s	143590	33.92%	1760s	**122336**	14.1%	0.90s	130996	22.18%	228s	123523	15.2%	0.90s
pcb442	58891	15.98%	897s	57114	12.48%	1760s	58197	14.6%	0.94s	57051	12.35%	228s	**55479**	9.26%	0.91s
Avg Gap		9.90%			7.63%			3.27%			5.14%			2.59%	

From the table, it can be observed that FEAM has a significant advantage compared to other DRL algorithms. The average gap between FEAM and the optimal length in the dataset is 2.59%, which is lower than other algorithms. This indicates that the proposed algorithm has better generalization compared to other algorithms. In terms of runtime, the time cost of FEAM is close to POMO and lower than other algorithms.

In order to compare the performance of these 5 algorithms on the public dataset more clearly, the number of optimal solutions found for each algorithm in 33 instances was statistically analyzed, and the results are shown in Fig. 4. From the graph, it can be observed that the AM+LCP algorithm failed to find the optimal solution, while the AM and DRL-2opt algorithms found fewer optimal solutions, with 1 and 4 respectively. The POMO algorithm found 12 optimal solutions. Our algorithm found 20 optimal solutions, more than the other algorithms.

These results indicate that compared to the comparative algorithms, our algorithm exhibits stronger solving performance, validating the effectiveness of the proposed improvement method.

4.4 Ablation Experiment

To validate the effectiveness of our proposed improvements, we conducted ablation experiments separately on the model structure and the inference method.

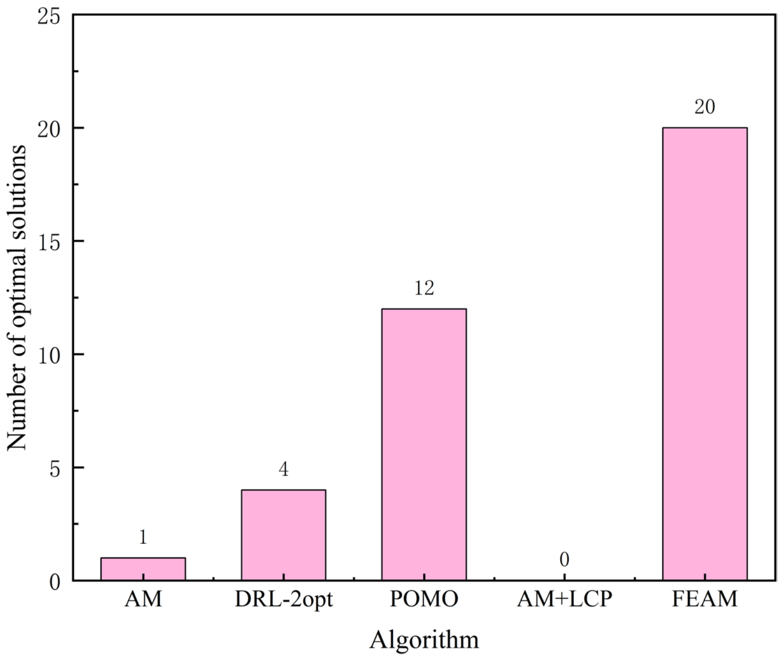

Fig. 4. The number of optimal solutions obtained by each algorithm

Model Structure Ablation Experiment. To validate the impact of the introduced feature selection layer and graph embedding layer on the model's performance, we conducted ablation experiments using TSP200, comparing FEAM with its three variants. Variant 1 does not use the graph aggregation layer, variant 2 does not use the feature selection layer, and variant 3 does not use both. The experimental results are shown in Table 6.

Table 6. Ablation experiments on key components of the model

Model	Len	Gap
Variant 1	10.825	0.98%
Variant 2	10.847	1.19%
Variant 3	11.021	2.81%
FEAM	**10.798**	**0.73%**

From the table, it can be observed that FEAM produces the best results with the smallest gap from the optimal solution. Applying the graph embedding layer or the feature selection layer individually also leads to improvements in performance, indicating that both components play a positive role in the algorithm.

Inference Method Ablation Experiment. To validate the effectiveness of our proposed rotation-based augmentation method and identify an appropriate rotation angle, we conducted ablation experiments using both TSP20 and TSP200 datasets. The results were compared between 8x augmentation and 16x augmentation. The results are shown in Fig. 5, where "8x" represents the 8x augmentation method, and "16xX" represents the 16x augmentation method with a rotation of X degrees.

From the graph, it can be observed that, compared to the 8x augmentation, the 16x augmentation with rotation yields results closer to the optimal solution, indicating the effectiveness of our proposed rotation augmentation method. Specifically, the best results are achieved when the rotation angle is 30°. However, as the rotation angle continues to increase, the results deteriorate. This may be because with a larger rotation angle, more nodes fall outside the training range, leading to a decrease in model performance. Additionally, it can be noticed that the variation in results for TSP20 is smaller than for TSP200 under different augmentation methods. This is likely because TSP20 has fewer nodes, and when rotated at different angles, the difference in the number of nodes exceeding the training range is not significant, resulting in solutions that are closer to each other.

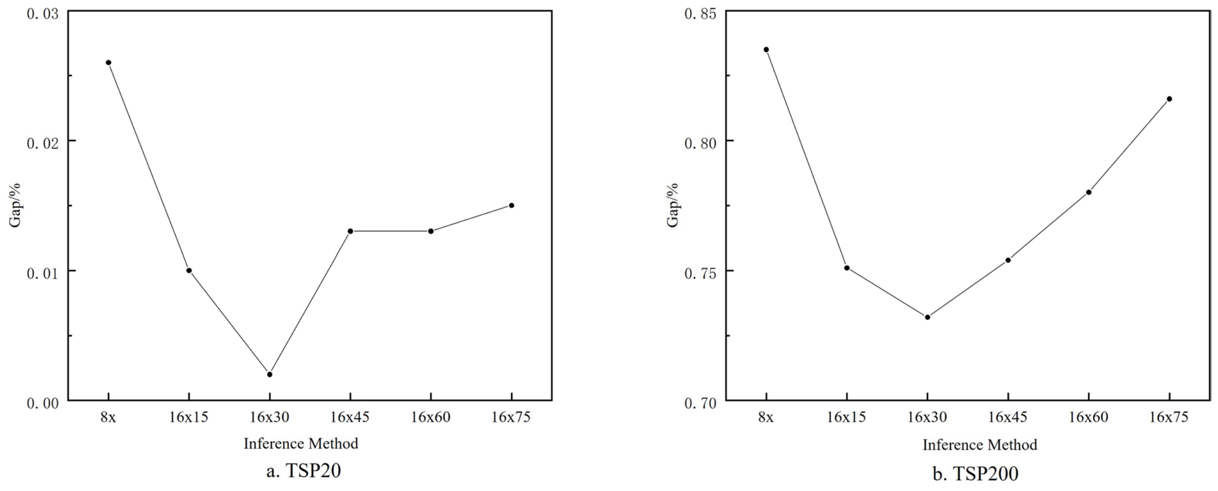

Fig. 5. Comparison of results from different inference methods

5 Conclusion

In this paper, we proposed a novel end-to-end DRL algorithm for solving the TSP. In terms of the model, we introduced the FEAM architecture, which incorporates a feature selection layer, a graph embedding layer, and a Transformer to better extract features of the nodes, resulting in richer and more accurate node representations and improved solving performance. Regarding the inference, we introduced the rotation augmentation inference method, allowing the model to consider the problem from various perspectives and thereby enhancing the quality of solutions. Through extensive experiments on both random TSP datasets and the TSPLIB dataset, our approach demonstrated superior performance compared to other well-known DRL methods, showing excellent generalization capabilities. This work provides new insights into real-time solving of vehicle routing problems. In future research, we will explore extending our approach to address more complex routing problems, such as vehicle routing problems with split deliveries and vehicle routing problems with time windows.

6 Acknowledgement

This research was funded by National Key Research and Development Program of China [grant numbers 2020YFB2103604].

References

[1] T. Zhang, Y.Q. Zhou, G. Zhou, W. Deng, Q.F. Luo, Discrete Mayfly Algorithm for spherical asymmetric traveling salesman problem, Expert Systems with Applications 221(2023) 119765.
[2] F. Su, L. Kong, H. Wang, Z. Wen, Modeling and application for rolling scheduling problem based on TSP, Applied Mathematics and Computation 407(2021) 126333.
[3] A. Madani, R. Batta, M. Karwan, The balancing traveling salesman problem: application to warehouse order picking, Top 29(2021) 442-469.
[4] S.H. Wang, Y.F. Liu, T.C. Wang, Y. Li, X.Y. Zhang, Exploring Object-Centric Temporal Modeling for Efficient Multi-View 3D Object Detection, in: Proc. 2023 IEEE International Conference on Computer Vision, 2023.
[5] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Proc. 2015 Conference and Workshop on Neural Information Processing Systems, 2015.
[6] C.K. Joshi, T. Laurent, X. Bresson, An efficient graph convolutional network technique for the travelling salesman problem, arXiv preprint arXiv:1906.01227 (2019).
[7] W. Kool, H.H. Van, J. Gromicho, M. Welling, Deep policy dynamic programming for vehicle routing problems, in: Proc. 2022 International conference on integration of constraint programming, artificial intelligence, and operations research, 2022.
[8] M. Nazari, A. Oroojlooy, L. Snyder, M. Takác, Reinforcement learning for solving the vehicle routing problem, in: Proc. 2018 Conference and Workshop on Neural Information Processing Systems, 2018.
[9] J. Zheng, K. He, J. Zhou, Y. Jin, C.M. Li, Combining reinforcement learning with Lin-Kernighan-Helsgaun algorithm for the traveling salesman problem, in: Proc. 2021 AAAI Conference on Artificial Intelligence, 2021.
[10] H. Lu, X. Zhang, S. Yang, A learning-based iterative method for solving vehicle routing problems, in: Proc. 2019 International Conference on Learning Representations, 2019.
[11] Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning improvement heuristics for solving routing problems, IEEE Transactions on Neural Networks and Learning Systems 33(9)(2021) 5057-5069.
[12] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, J. Ye, Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach, in: Proc. 2018 ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2018.
[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proc. 2017 Conference and Workshop on Neural Information Processing Systems, 2017.
[14] X. Chen, Y. Tian, Learning to perform local rewriting for combinatorial optimization, in: Proc. 2019 Conference and Workshop on Neural Information Processing Systems, 2019.
[15] L. Gao, M. Chen, Q. Chen, G. Luo, N. Zhu, Z. Liu, Learn to design the heuristics for vehicle routing problem. <https://arxiv.org/abs/2002.08539>, 2020 (accessed 28.12.2023).
[16] L. Xin, W. Song, Z. Cao, J. Zhang, Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem, Advances in Neural Information Processing Systems 34(2021) 7472-7483.
[17] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning. <https://arxiv.org/abs/1611.09940>, 2016 (accessed 28.12.2023).
[18] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, L. M. Rousseau, Learning heuristics for the tsp by policy gradient, in: Proc. Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, 2018.
[19] W. Kool, H.H. Van, M. Welling, Attention, Learn to Solve Routing Problems!, in: Proc. 2019 International Conference on Learning Representations, 2019.
[20] A. Nowak, S. Villar, A.S. Bandeira, J. Bruna, A note on learning algorithms for quadratic assignment with graph neural networks, in: Proc. 2017 International Conference on Machine Learning, 2017.
[21] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in: Proc. 2017 Conference and Workshop on Neural Information Processing Systems, 2017.
[22] S. Woo, J. Park, J.Y. Lee, I.S. Kweon, Cbam: Convolutional block attention module, in: Proc. 2018 European Conference on Computer Vision, 2018.
[23] K. Lei, P. Guo, Q.X. Wang, W.C. Zhao, L.S. Tang, End-to-end deep reinforcement learning framework for multi-depot vehicle routing problem, Application Research of Computers 39(10)(2022) 3013-3019.
[24] Y.D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, S. Min, Pomo: Policy optimization with multiple optima for reinforcement learning, in: Proc. 2020 Conference and Workshop on Neural Information Processing Systems, 2020.
[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: Proc. 2018

International Conference on Learning Representations, 2018.
[26] K.M. He, X.Y. Zhang, S.Q. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition, 2016.
[27] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine learning 8(1992) 229-256.
[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J.J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Proc. 2019 Conference and Workshop on Neural Information Processing Systems, 2019.
[29] M. Kim, J. Park, Learning collaborative policies to solve NP-hard routing problems, in: Proc. 2021 Conference and Workshop on Neural Information Processing Systems, 2021.
[30] K. Helsgaun, An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems, Roskilde: Roskilde University 12(2017).
[31] M. Jung, J. Lee, J. Kim, A Lightweight CNN-Transformer Model for Learning Traveling Salesman Problems. <https://arxiv.org/abs/2305.01883>, 2023 (accessed 02.01.2024).
[32] P.R.d.O. Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning, in: Proc. 2020 Asian conference on machine learning, 2020.
[33] Z.H. Fu, K.B. Qiu, H. Zha, Generalize a small pre-trained model to arbitrarily large TSP instances, in: Proc. 2021 AAAI Conference on Artificial Intelligence, 2021.