Solving Traveling Salesman Problem with Deep Reinforcement Learning and Knowledge Distillation

Xiaowen Li, Xiaofeng Gao, Shaoyao Niu, Wenxuan He, Wanru Gao, and Qidong Liu(✉)

Zhengzhou University, Zhengzhou, China
{xiaowli,ugaoxf218,ieniusy,iewxhe}@gs.zzu.edu.cn, {iewrgao,ieqdliu}@zzu.edu.cn

Abstract. The Traveling Salesman Problem (TSP) is a renowned combinatorial optimization problem with wide-ranging practical applications. However, TSP is an NP-hard problem, which makes finding an efficient and accurate solution computationally challenging. While deep learning and reinforcement learning have shown promise in solving TSP, prevailing methods suffer from some limitations, such as excessive model complexity and long inference durations. In this work, we propose a method for solving TSP, which extracts knowledge from a complex teacher model to a lightweight student model. The teacher model adopts an encoder-decoder framework and uses the mixed chunk attention mechanism to extract the features of the input city sequence. The student model adopts the same architecture as the teacher model, but simplifies network parameters by decreasing the dimensionality of hidden layers via knowledge distillation. Instead of using a tour generated by the teacher model to guide the student model, our teacher model guides the student model at each time step. Extensive experiments demonstrate the competitive and effective nature of our model.

Keywords: traveling salesman problem (TSP) · knowledge distillation · reinforcement learning

1 Introduction

The Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization [1] which has extensive applications in various fields, such as transportation, logistics, tourism, infrastructure development, and so on [2]. The TSP aims to find the shortest route that visits each city and returns to the starting city in a given set of cities.

Supported in part by the National Natural Science Foundation of China under Grant 62276238, in part by the Outstanding Youth Foundation of He'nan Scientific Committee under Grant 232300421095, and in part by the China Postdoctoral Science Foundation under Grant 2022T150590 and Grant 2020M672275.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
V. C. M. Leung et al. (Eds.): Qshine 2023, LNICST 574, pp. 360–374, 2024.
https://doi.org/10.1007/978-3-031-65123-6_26

The traditional algorithms for solving TSP can be classified into three main categories [3]. The first category is exact algorithms, which search the complete solution space to find the optimal solution that satisfies the constraints [4]. However, due to the NP-hard nature of TSP, it becomes more and more difficult to find the theoretical optimal solution as the problem size grows. Therefore, these methods are only suitable for small-scale tasks. The second category is approximation algorithms, which provide faster solutions [5] but susceptible to getting stuck in local optima. As a result, the obtained solutions are typically suboptimal or approximate solutions to the global optimum. The third category is heuristic algorithms, primarily inspired by nature-based algorithms such as genetic algorithms [6], particle swarm optimization [7], and ant colony optimization [8]. These heuristic algorithms exhibit relatively high computational efficiency. However, the design of these heuristic rules [9,10] is typically based on manual expert knowledge, resulting in increased computation time and cost as the problem size grows.

Traditional algorithms has complex modeling process and exhibit lower efficiency for solving TSP. With the continuous development of artificial intelligence technologies such as deep learning and reinforcement learning, as well as their excellent performance in various problems, researchers have begun to explore end-to-end methods for solving TSP [11–14]. Compared to traditional algorithms, deep reinforcement learning-based methods offer simpler modeling and higher efficiency, which demonstrate significant potential and advantages in solving TSP. Consequently, deep reinforcement learning methods have become one of the mainstream choices for solving TSP.

However, current approaches based on deep reinforcement learning for solving TSP commonly suffer from the issues of large parameter sizes and long inference time. Therefore, this work introduces the technique of knowledge distillation into the modeling framework for solving TSP. Knowledge distillation [15] is a technique that transfers knowledge from large and complex models (referred to the teacher model) to smaller models being trained (referred to the student model). Specifically, the core idea behind knowledge distillation is to enable the student model to imitate the behavior of the teacher model under its supervision, resulting in the student model exhibiting more competitive performance. Teacher models usually have excellent performance and generalization ability, but are often complex and have many parameters. In contrast, the network size of the student model is smaller, which can meet the constrains of hardware resources.

Currently, the existing method has used knowledge distillation to solve TSP. Literature [16] guides the student model based on tours, where the teacher model generates complete paths one time and employs them to guide the training of the student model. However, if the student model makes errors, the teacher model cannot promptly correct the student model, which can affect the learning speed of the student model to some extent. To address above issue, we propose an novel model, where the teacher model guides the student model at every time step. Specifically, we first pre-train the teacher model. Then, at each time step, we measure the difference between the probability distributions outputted by

the teacher model and the student model. And the nodes selected by the teacher model are used to update their states. So in our model, the student model can receive timely guidance from the teacher model.

Our main contributions are summarized as follows:

(1) We propose a new knowledge distillation framework for solving TSP. In this framework, the teacher model guides the student model at each time step, which can alleviate the problem of error accumulation and accelerate the model's training speed, ultimately yielding a lightweight student model.
(2) We introduce the mixed chunk attention mechanism in the encoder of the teacher model, which reduces the time complexity of the encoding process from $O(n^2)$ to linear, thereby reducing the inference time.
(3) Through experiments conducted on synthetic datasets and real-world datasets, we demonstrate the effectiveness of teacher model-guided. The proposed KDRL model achieves a significant reduction in parameter size and inference time while maintaining a minimal loss in accuracy.

2 Proposed Method

In this section, we first introduce the mathematical formulation of TSP, then detail the teacher model, the student model and the loss function in our method.

2.1 Mathematical Formulation

The TSP problem instance s is typically represented as an undirected complete graph with n nodes [13]. The features of node $i \in (1, \cdots, n)$ are denoted by its two-dimensional coordinates X_i. The solution is a permutation of nodes (tour) obtained through the model, represented as $\pi = (\pi_1, \cdots, \pi_n)$. Here $\pi_t \in (1, \cdots, n) \wedge \{\pi_t \neq \pi_{t'}, \forall t \neq t'\}$.The solution is measured by the length of the permutation $Len(\pi)$, given by $Len(\pi) = ||X_{\pi_n} - X_{\pi_1}||_2 + \sum_{i=1}^{n-1} ||X_{\pi_i} - X_{\pi_{i+1}}||_2$. The total reward of a solution permutation is defined as the negative of its length, i.e., $-Len(\pi|s)$. The policy π to be learned in our method is modeled as:

$$p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{1:t-1}), \tag{1}$$

where $p(\pi|s)$ is the probability of obtaining the solution permutation π given the problem instance s, and θ is the trainable parameters of the policy network.

2.2 Teacher Model

Since the TSP involves mapping a random sequence of the input nodes to a specific sequence of the nodes in the solution permutation, it can be viewed as a Seq2Seq task. In this work, we adopt the classical encoder-decoder framework to design the teacher model. As shown in Fig. 1, our teacher model consists of

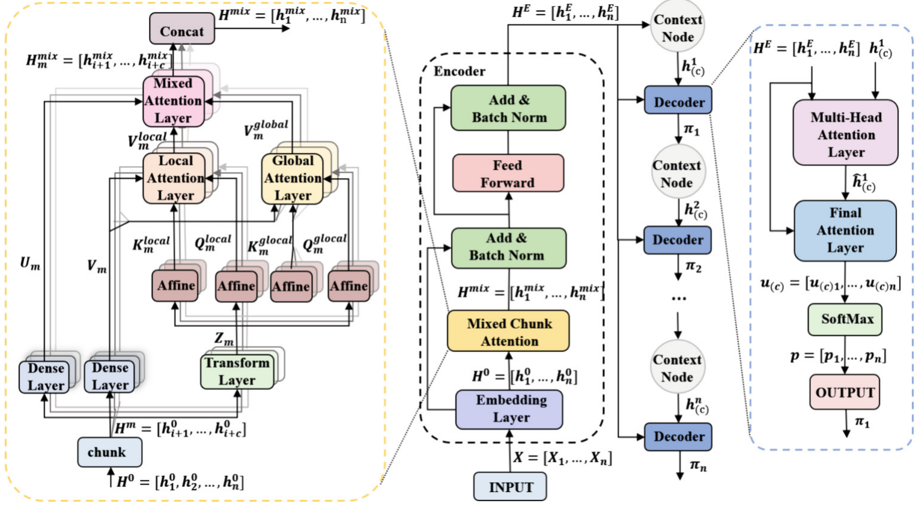


Fig. 1. The architecture of the teacher model.

encoder, context node and decoder. Specifically, the encoder of the teacher model takes the two-dimensional coordinates of the nodes $X = [X_1, X_2, \cdots, X_n]$ as input, and obtains static features $H^E = [h_1^E, h_2^E, \cdots, h_n^E]$. The static features are used to construct context node $h_{(c)}^1$. Then the static features and context node are input to the decoder. The decoder calculates the probability distribution of the candidate nodes $p = [p_1, p_2, \cdots, p_n]$, and then selects the node π_1 according to the probability distribution. Next, in a loop, the context node $h_{(c)}^t$ is updated based on the node selected in the previous time step, i.e. π_{t-1}. And the next node π_t is chosen according to the context node $h_{(c)}^t$ and the static features of the candidate nodes until a complete permutation solution is obtained.

Encoder. The encoder of the teacher model resembles the encoder of the Transformer. However, since the multi-head attention mechanism has the time complexity of $O(n^2 d)$ during similarity calculation, we adopt the mixed chunk attention mechanism proposed by Hua et al. [17] as an alternative. The mixed chunk attention combine the Gated Linear Units (GLU) [18] and Linear Attention mechanism to reduce the time complexity to linear.

Specifically, the encoder takes the two-dimensional coordinates of nodes in the problem instance, represented as $X = [X_1, X_2, \cdots, X_n]$, as its input, and $X \in \mathbb{R}^{n \times d_x}$. Then the input data enters the initial embedding layer to extract the initial feature. The computation formula for the embedding layer is as follows:

$$h_i^0 = x_i W_e + b_e, \tag{2}$$

where $W_e \in \mathbb{R}^{d_x \times d_h}$ is the weight matrix. b_e represents the bias. The initial feature of all nodes represents as $H^0 = [h_1^0, h_2^0, \cdots, h_n^0]$. When initial feature enters

the mixed chunk layer, it is divided into M non-overlapping blocks, each with a length of c. For the m-th block of data H_m, it simultaneously passes through two Dense layers with Swish activation functions to obtain $U_m \in \mathbb{R}^{c\times d_e}$ and $V_m \in \mathbb{R}^{c\times d_e}$, where d_e is the dimension of the intermediate expansion layer, and passes through a Transform layer to obtain $Z_m \in \mathbb{R}^{c\times d_s}$, where d_s represents the dimensionality during attention computation. Then, Z_m undergoes four simple affine transformations to obtain K_m^{local}, Q_m^{local}, K_m^{global} and Q_m^{global}. In order to realize the interaction and fusion of data within each block, it is necessary to calculate local attention. Its calculation formula is as follows:

$$V_m^{local} = relu^2(Q_m^{local} {K_m^{local}}^T + b_m)V_m, \tag{3}$$

where b_m is the bias. Assuming that the vector dimension of the input node is d, the time complexity of local attention is $O(M \times c^2 \times d) = O(ncd)$. So the time complexity of computing local attention is linear. Then the global attention of inter-block data is calculated. The calculation formula is as follows:

$$V_m^{global} = Q_m^{global}(\sum_{h=1}^{M} {K_h^{global}}^T V_h). \tag{4}$$

The order of multiplication in the attention matrix is redesigned. It is equivalent to the calculation formula of the linear attention mechanism [19]. Then, the outputs of local attention and global attention are fed into a mixed attention layer. The computational form of the mixed attention layer is similar to that of GLU. The difference is that the interconnection between input data cannot be obtained in GLU. However, the inputs of the mixed attention layer, which are the local attention and global attention, have already fused the interaction information between data. Its calculation formula is as follows:

$$O_m = [U_m \odot (V_m^{local} + V_m^{global})]W_o, \tag{5}$$

where $W_o \in \mathbb{R}^{d_e\times d_h}$. Finally, the mixed attention of M blocks is concatenated together to obtain the mixed attention of the whole instance, denoted as $H^{mix} = [h_1^{mix}, h_2^{mix}, ..., h_n^{mix}]$ and $H^{mix} \in \mathbb{R}^{n\times d_h}$. Therefore, it can be seen that the time complexity can be reduced from $O(n^2)$ to linear. The resulting mixed chunk attention is subjected to skip connection and batch normalization in the residual network. Then it is fed into the feed-forward layer, followed by a skip connection and batch normalization. Finally, the static features extracted by the encoder are expressed as $H^E = [h_1^E, h_2^E, ..., h_n^E]$.

Context Node. The context node can incorporate key node information in the decoding process to improve the quality of decoding. At the t-th time step, the context node $h_{(c)}^t \in \mathbb{R}^{3\times d_h}$ includes the static feature of the initial node $h_{\pi_1}^E \in \mathbb{R}^{1\times d_h}$, the static feature of the selected node in the previous time step $h_{\pi_{t-1}}^E \in \mathbb{R}^{1\times d_h}$ and graph embedding information $h_{(g)} \in \mathbb{R}^{1\times d_h}$ of all nodes. $h_{(g)}$ is calculated as follows:

$$h_{(g)} = \frac{1}{n}\sum_{i=1}^{n} h_i^E. \tag{6}$$

If $t = 1$, the information of the initial node and the selected node in the previous time step are learnable parameters. The context node needs to be updated at every time step.

Decoder. Our decoder is similar to the AM (Attention Model) [13]. At the t-th time step, the decoder first applies the multi-head attention layer to the context node and the static feature of the candidate nodes to extract the relevant features. And use a mask matrix to control all nodes to be selected only once. The query vector of this layer comes from the context node, i.e., $q = h_{(c)}W_Q$. The key vector and the value vector are obtained by embedding candidate nodes, i.e., $k_i = h_i^E W_K$,$v_i = h_i^E W_V$. $W_Q \in \mathbb{R}^{d_h \times 3d_h}$, $W_K \in \mathbb{R}^{d_h \times d_h}$ and $W_V \in \mathbb{R}^{d_h \times d_h}$ are the trainable weight matrixs. The calculation formula is as follows:

$$\hat{h}_{(c)}^t = \begin{cases} softmax(\frac{q^T k_j}{\sqrt{d_k}})v_j, & if\ j \neq \pi_{t'}, \forall t' < t \\ -\infty, & otherwise \end{cases}, \tag{7}$$

where $d_k = d_h$. Then enter the single-head attention layer to calculate the final correlation value. The query vector and the key vector are similar to the multi-head attention layer. The calculation formula is as follow:

$$q_{(c)} = \hat{h}_{(c)}^t W_F, \tag{8}$$

$$k_i^f = h_i^E W_M, \tag{9}$$

$$u_{(c)j} = \begin{cases} C \cdot tanh(\frac{q_{(c)}^T k_j^f}{\sqrt{d_k}}), & if\ j \neq \pi_{t'}, \forall t' < t \\ -\infty, & otherwise \end{cases}, \tag{10}$$

where $W_F \in \mathbb{R}^{d_h \times d_h}$ and $W_M \in \mathbb{R}^{d_h \times d_h}$ are the trainable weight matrixs. C is a constant. During the experiments, it's 10. Finally, the probability of the candidate nodes are calculated through the SoftMax function and then select the next node. The calculation formula is as follows:

$$p_i = p_\theta(\pi_t = i | s, \pi_{1:t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}}. \tag{11}$$

Training. In order to mitigate the challenges associated with obtaining costly and time-consuming labels for training data in TSP, we use RL to train the teacher model. The quality of the current policy is evaluated by the total length of the solution permutation. Consequently, for a given problem instance s, the loss function is defined as follows:

$$L_{Te}(\theta^{Te}|s) = E_{p_{\theta^{Te}}(\pi^{Te}|s)}[Len(\pi^{Te})], \tag{12}$$

where θ^{Te} and π^{Te} respectively denote the model's parameters and solution permutation. $Len(\pi^{Te})$ represents the total length of the solution permutation.

We optimize the loss function using the REINFORCE algorithm combined with a Baseline [20]. Therefore, the gradient computation formula for the teacher model is as follows:

$$\nabla L_{Te}(\theta^{Te}|s) = E_{p_{\theta^{Te}}(\pi^{Te}|s)}[(Len(\pi^{Te}) - b(s))\nabla log p_{\theta^{Te}}(\pi^{Te}|s)], \tag{13}$$

where $b(s)$ represents the baseline, which can reduce variance. In the experimental operation, the choice of baseline has a great impact on model performance. We select the Greedy Rollout strategy as the baseline, because it's simple yet effective. Additionally, we employ the Adam optimizer to update the gradients during training.

2.3 Student Model

We employ the pre-trained teacher model to provide guidance to the student model named Lightweight Deep Reinforcement Learning Model Based on Knowledge Distillation (KDRL). During the guidance process, the teacher model only predicts the next node based on the current state at each time step, no longer updates its parameters. The student model shares a similar architecture to the teacher model but with reduced network parameters to enhance inference speed. Specifically, we decrease the dimension of the student model's hidden layers to 32, while the teacher model's hidden layer dimension is 128. Figure 2 illustrates the comparison of feature tensor sizes between the teacher model and student model at every layer within a single time step for the 50-node instance. The batch size is set to 512.

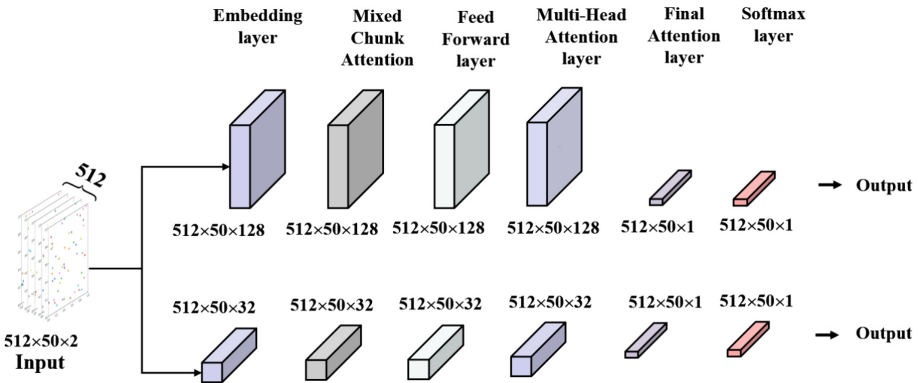


Fig. 2. Tensor size comparison between the teacher model and the student model within a single time step.

2.4 Loss Function

Similar to the typical knowledge distillation models, the loss function for training the KDRL model comprises two components: the student loss (L_{St}) and the distillation loss (L_{Di}). The loss function is defined as $L = \alpha L_{St} + \beta L_{Di}$, where L_{St} represents the student model's loss and L_{Di} represents the distillation loss. The hyperparameters α and β are used to balance the importance of the two losses, so that the model can achieve better performance.

Student Loss. The student model is trained using the same method as the teacher model, which is reinforcement learning. Similarly, we measure the performance of the student model based on the total length of the solution permutation generated by the student model. The loss function of the student model, considering a specific problem instance s, is calculated as follows:

$$L_{St}(\theta^{St}|s) = E_{p_{\theta^{St}}(\pi^{St}|s)}[Len(\pi^{St})], \tag{14}$$

where θ^{St} denotes the parameters of the student model. π^{St} represents the solution permutation obtained by the teacher model, and $Len(\pi^{St})$ represents the total length of the solution permutation.

Similarly, we employ the REINFORCE algorithm combined with Greedy Rollout strategy as the baseline to estimate the gradient of the loss function. The gradient computation formula of the student model is as follows:

$$\nabla L_{St}(\theta^{St}|s) = E_{p_{\theta^{St}}(\pi^{St}|s)}[(Len(\pi^{St}) - b(s))\nabla log p_{\theta^{St}}(\pi^{St}|s)]. \tag{15}$$

Distillation Loss. The distillation loss is used to measure the discrepancy between the action probability distributions outputted by the teacher model and the student model, thereby encouraging the student model to emulate the teacher model's node selection. The computation process is illustrated in Fig. 3. At the initial time step, the node information is inputted into the two models. In subsequent time steps, we calculate the discrepancy of their probability distributions. Next, we sample the next action, i.e., select the next node, according to the probability distribution of the teacher model. We use this action to respectively update their next states. Thus, at each time step, the teacher model provides guidance to the student model. In detail, the two models respectively output their logits, which are processed through a softmax function with a temperature parameter T which is used to adjust the smoothness of the predicted distribution. The probability distribution calculation formula for the teacher model at time step t is as follows:

$$p_{\theta^{Te}}(\pi_t^{Te} = i|s^{Te}, \pi_{1:t-1}^{Te}) = \frac{exp(u_{(c)i}^{Te}/T)}{\sum_j exp(u_{(c)j}^{Te}/T)}, \tag{16}$$

where θ^{Te} represents the pre-trained network parameters of the teacher model. π_t^{Te} represents the node selected by the teacher model at the t-th time step. s^{Te}

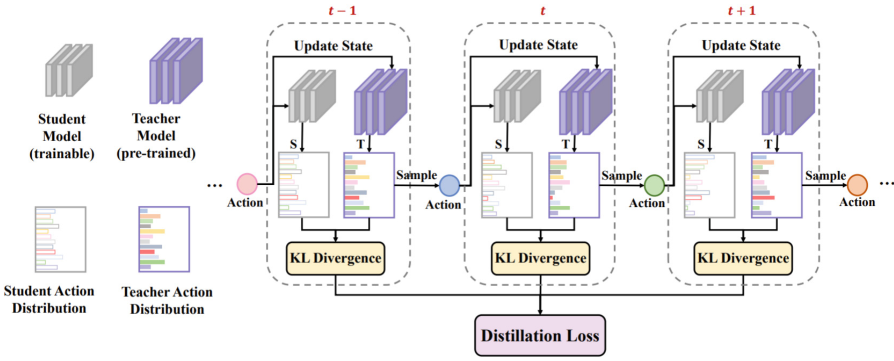


Fig. 3. The distillation loss is calculated by comparing the probability distributions of the teacher model and the student model at each time step. Both models update their states based on the actions generated by the teacher model.

represents the state of the teacher model at current time step. $\pi^{Te}_{1:t-1}$ represents the solution permutation generated by the teacher model up to time step $(t-1)$. $u^{Te}_{(c)i}$ denotes the compatibility value of the i-th node in the teacher model's decoder. T represents the temperature parameter.

Similarly, the probability distribution calculation formula for the student model at time step t is as follows:

$$p_{\theta^{St}}(\pi^{St}_t = i|s^{St}, \pi^{Te}_{1:t-1}) = \frac{exp(u^{St}_{(c)i}/T)}{\sum_j exp(u^{St}_{(c)j}/T)}, \tag{17}$$

where θ^{St} represents the network parameters of the student model. π^{St}_t represents the node selected by the student model at the t-th time step. s^{St} represents the state of the student model at current time step. $u^{St}_{(c)i}$ denotes the compatibility value of the i-th node in the student model's decoder.

The distillation loss measures the discrepancy of the probability distributions obtained by the teacher model and the student model at all time steps. At each time step, the calculation formula for the distillation loss is as follows:

$$L_{Di} = \sum_{t=1}^{T} \Psi[p_{\theta^{Te}}(\pi^{Te}_t = i|s^{Te}, \pi^{Te}_{1:t-1}), p_{\theta^{St}}(\pi^{St}_t = i|s^{St}, \pi^{Te}_{1:t-1})], \tag{18}$$

where the $\Psi(\cdot)$ is used to measure the distribution divergence between the two models. In this model, we utilize the Kullback-Leibler Divergence (KL divergence) [21] as the measure, expressed as follows:

$$KL(p_{\theta^{Te}}||p_{\theta^{St}}) = \sum_i p_{\theta^{Te}}(logp_{\theta^{Te}} - logp_{\theta^{St}}). \tag{19}$$

3 Experiments

3.1 Experiment Settings

The datasets used in our experiments include synthetic datasets and real-world datasets. For the synthetic datasets, the nodes are generated by uniformly sampling random 2D coordinates within the unit square $[0,1] \times [0,1]$, representing the problem instances. The real-world datasets are obtained from the widely-used TSPLIB dataset [22]. All experiments are conducted on a Linux-based Ubuntu system, utilizing GPUs for training and testing. The experiments are performed using a Geforce RTX 3080 graphics card.

During training, we set the epochs of the teacher model to 200 and the epoch of KDRL to 40. Each epoch processes 256 batches of data, with a batch size of 512. The learning rate is set to 0.0001. After each epoch, the model is evaluated on 1000 problem instances. If the updated weight parameters shown an improvement of over 0.05, the Baseline is updated, replacing the previous weight parameters. The parameter α, β and T are respectively set to 0.5, 5.0 and 2. During testing, the two models employ two different strategies, greedy strategy and sampling strategy, to select the next node based on the probability distributions of candidate nodes.

Table 1. Comparison of the KDRL model with the teacher model on synthetic datasets.

Model		Concorde	Teacher (Greedy)	Teacher (Sampling)	KDRL (Greedy)	KDRL (Sampling)
n = 20	Length	3.84	3.85	3.83	3.86	3.84
	Gap	0.00%	0.18%	0.05%	0.42%	0.05%
	Time	52 s	0.3 s	283 s	0.2 s	246 s
	Memory Usage	-	2487MiB	2487MiB	2119MiB	2119MiB
n = 50	Length	5.70	5.78	5.72	5.87	5.73
	Gap	0.00%	1.40%	0.35%	2.98%	0.53%
	Time	59 s	0.8 s	713 s	0.6 s	603 s
	Memory Usage	-	4597MiB	4597MiB	3231MiB	3231MiB
n = 100	Length	7.76	8.06	7.88	8.29	7.97
	Gap	0.00%	3.86%	1.54%	6.83%	2.71%
	Time	93 s	1.8 s	1898s	1.3 s	1393 s
	Memory Usage	-	6045MiB	6045MiB	4033MiB	4033MiB

3.2 Comparison Experiments

To better demonstrate the lightweight nature of KDRL, we compare it with the teacher model using evaluation metrics including Length, Gap, and Time. Length refers to the total length of the solution permutation generated by the model. Gap is used to evaluate the gap between current solution and the optimal solution. Time refers to the time that the model needs to generate a solution. Additionally, we introduce "Memory Usage" to provide a more intuitive representation of the model's lightweight characteristics. For other methods, we do

not measure the memory usage, so we use "-" to indicate this. Tabel 1 and Tabel 2 respectively present the experimental results of the teacher model and KDRL on both the synthetic datasets and the real-world datasets. Concorde [23] is a professional solver widely recognized for achieving the world's best records on many TSP datasets. From the results, it can be observed that compared to the teacher mode, the KDRL slightly reduces the quality of the solutions but significantly reduces memory usage and inference time. To further validate the effectiveness of the KDRL, we also compare it with several classical baseline models.

Table 2. Comparison of the KDRL model with the teacher model on real-world datasets.

TSPLIB		eil51	berlin52	st70	eil76	rat99	kroA100	kroB100	eil101
Concorde	Length	426	7542	675	538	1211	21282	22142	629
	Gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Time	0.02 s	0.04 s	0.05 s	0.03 s	0.12 s	0.09 s	0.20 s	0.07 s
Teacher(Sampling)	Length	426	8229	682	553	1430	25002	23952	655
	Gap	0.00%	9.11%	1.04%	2.79%	18.08%	17.48%	8.17%	3.53%
	Time	0.08 s	0.08 s	0.12 s	0.14 s	0.19 s	0.19 s	0.19 s	0.19 s
KDRL(Sampling)	Length	426	7735	682	555	1461	24805	25095	669
	Gap	0.00%	2.56%	1.04%	3.16%	20.64%	16.55%	13.34%	6.36%
	Time	0.06 s	0.06 s	0.08 s	0.09 s	0.12 s	0.13 s	0.13 s	0.13 s

In addition to Concorde mentioned above, the baselines we use include:

- LKH3 [24]: The LKH (Lin-Kernighan-Helsgaun) algorithm, a local optimization algorithm based on λ-opt exchanges, is an effective heuristic algorithm for TSP.
- OR-Tools: An open-source solver developed by Google in 2016 for solving combinatorial optimization problems, providing various interfaces.
- Bello et al. [11]: The first deep reinforcement learning model that applies RL to solve the TSP problem.
- Dai et al. [25]: This method treats the input of the TSP problem as a graph and utilizes graph neural networks to extract information from the graph.
- Kool et al. [13]: Based on the Transformer, the proposed AM (Attention Model) combines attention mechanisms for solving TSP.
- Joshi et al. [26]: This method use a combination of GNN and LSTM to encodes the input data. It is trained by supervised learning and RL.
- Lou et al. [27]: This method use GCN to encode the input data. Additionally, it update the information in different problem instances of various sizes using the same number of message steps by controlling the number of neighbors for each node.

The experimental results comparing the KDRL model with the baseline models are shown in the Tabel 3. It can be observed that the KDRL model using the Greedy strategy achieves improvement in terms of inference time. Although the KDRL model using the Sampling strategy performs better than the Greedy strategy in terms of solution quality, it leads to increased inference time.

Table 3. Comparison of the KDRL model with the baselines on synthetic datasets.

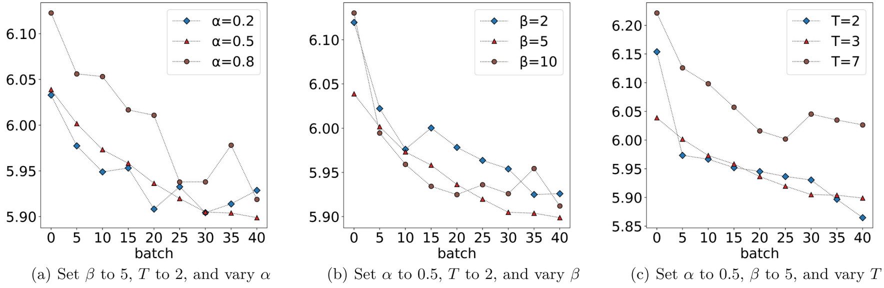
Model	n = 20			n = 50			n = 100		
	Length	Gap	Time	Length	Gap	Time	Length	Gap	Time
Concorde	3.84	0.00%	52 s	5.70	0.00%	59 s	7.76	0.00%	93 s
LKH3	3.84	0.00%	16 s	5.70	0.00%	147 s	7.76	0.00%	650 s
OR-Tools	3.85	0.37%	52 s	5.80	1.83%	147 s	7.99	2.90%	712 s
Bello	3.89	1.42%	-	5.95	4.46%	-	8.30	6.90%	-
Dai	3.89	1.42%	-	5.99	5.16%	-	8.31	7.03%	-
Kool(Greedy)	3.85	0.34%	0.3 s	5.80	1.76%	0.7 s	8.12	4.53%	1.7 s
Kool(Sampling)	3.84	0.08%	260 s	5.73	0.52%	704 s	7.94	2.26%	1858s
Joshi	3.86	0.06%	5 s	5.87	3.10%	27 s	8.14	8.38%	93 s
Luo	3.85	0.38%	1 s	5.73	0.65%	9 s	8.04	3.61%	2 s
KDRL(Greedy)	3.86	0.42%	0.2 s	5.87	2.98%	0.6 s	8.29	6.83%	1.3 s
KDRL(Sampling)	3.84	0.05%	246 s	5.73	0.53%	603 s	7.97	2.71%	1393 s

3.3 Ablation Experiments

To validate the validity and reliability of KDRL, we conduct ablation experiments. We separated the distillation loss and the student loss in the KDRL to form two variants, the DL-Free Model and the SL-Free Model. Due to hardware limitations, the batch size is 256 when conducting experiments on the synthetic dataset with problem instance size of 100. Table 4 shows the comparison between the original model and its varients on different node sizes. The results show that KDRL outperforms both the DL-Free Model and the SL-Free Model on all three metrics. Therefore, combining the distillation loss with the student loss enhances the effectiveness of the model. The combined loss function improves the solution quality while reducing the solution time.

Table 4. Ablation study of KDRL loss function.

Model		Concorde	DL-Free (Greedy)	DL-Free (Sampling)	SL-Free (Greedy)	SL-Free (Sampling)	KDRL (Greedy)	KDRL (Sampling)
n = 20	Length	3.84	3.87	3.84	3.86	3.84	3.86	3.84
	Gap	0.00%	0.78%	0.09%	0.43%	0.06%	0.42%	0.05%
	Time	52 s	0.2 s	247 s	0.2 s	251 s	0.2 s	246 s
n = 50	Length	5.70	5.95	5.78	5.90	5.74	5.87	5.73
	Gap	0.00%	4.39%	1.40%	3.51%	0.70%	2.98%	0.53%
	Time	59 s	0.6 s	615 s	0.6 s	604 s	0.6 s	603 s
n = 100	Length	7.76	8.40	8.14	8.30	7.98	8.29	7.97
	Gap	0.00%	8.25%	4.90%	6.96%	2.84%	6.83%	2.71%
	Time	93 s	1.3 s	1395 s	1.3 s	1401 s	1.3 s	1393 s



(a) Set β to 5, T to 2, and vary α (b) Set α to 0.5, T to 2, and vary β (c) Set α to 0.5, β to 5, and vary T

Fig. 4. Sensitivity analysis of hyperparameters affecting model performance in problem instances with 50 nodes.

3.4 Sensitivity Experiments of Hyperparameters

We conduct experiments on three important hyperparameters, α, β and T, which influence the performance of KDRL. The experiments are performed on a synthetic dataset with a problem size of 50 nodes. In each experiment, we vary only one parameter. The results are shown in Fig. 4. It can be observed that the KDRL model is most sensitive to the variation of T and least sensitive to the variation of the β. If α is set to 0.5, the model exhibits the fastest convergence rate and achieves the best experimental results. Setting β to 5 yields the best overall performance of KDRL. If T is set to 2, the model achieves the best performance. Therefore, based on the parameter sensitivity experiments, the optimal parameter settings are $\alpha = 0.5$, $\beta = 5$, and $T = 2$.

4 Conclusion

In this work, we propose a lightweight deep reinforcement learning model based on knowledge distillation (KDRL) for solving TSP. In the encoder, we adopt the mixed chunk attention mechanism to extract features of input data, which reduce time complexity. At each time step in the knowledge distillation process, the teacher model and KDRL respectively output the probability distributions of candidate nodes, and we use KL divergence to measure the difference between them. Meanwhile, at each time step, the selected node of the teacher model is used to update the states of the two models, ensuring that the teacher model guides the student model at each time step. Extensive experimental results demonstrate that our teacher model effectively guides the student model. The KDRL model maintains superior solution time and memory usage performance with a small solution quality loss. Future work will explore using multiple teacher models to simultaneously guide the student model to allow the student model to learn more. In addition, this work focuses on solving the classic symmetric TSP problem, and in the future, we can study TSP problem variants in different scenarios.

References

1. Papadimitriou, C.H.: The euclidean travelling salesman problem is NP-complete. Theor. Comput. Sci. **4**(3), 237–244 (1977)
2. Li, W., et al.: Parameterized algorithms of fundamental NP-hard problems: a survey. HCIS **10**, 1–24 (2020)
3. Gutin, G., Punnen, A.P.: The Traveling Salesman Problem and Its Variations, vol. 12. Springer, New York (2006). https://doi.org/10.1007/b101971
4. Woeginger, G.J.: Exact algorithms for NP-hard problems: a survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization — Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36478-1_17
5. Arora, S.: The approximability of NP-hard problems. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 337–348 (1998)
6. Yu, Y., Chen, Y., Li, T.: A new design of genetic algorithm for solving TSP. In: 2011 Fourth International Joint Conference on Computational Sciences and Optimization, pp. 309–313. IEEE (2011)
7. He, J., Shi, D., Wang, L.: An adaptive discrete particle swarm optimization for TSP problem. In: 2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA), pp. 393–396, vol. 2. IEEE (2009)
8. Junjie, P., Dingwei, W.: An ant colony optimization algorithm for multiple travelling salesman problem. In: First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC 2006), pp. 210–213, vol. 1. IEEE (2006)
9. Changyou, W., Xisong, F.: An agglomerative greedy brain storm optimization algorithm for solving the TSP. IEEE Access **8**, 201606–201621 (2020)
10. Skinderowicz, R.: Improving ant colony optimization efficiency for solving large TSP instances. Appl. Soft Comput. **120**, 108653 (2022)
11. Bello, I., et al.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
12. Nazari, M., et al.: Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
13. Dauphin, Y.N., et al.: Language modeling with gated convolutional networks. In: International Conference on Machine Learning, pp. 933–941. PMLR (2017)
14. Choromanski, K., et al.: Rethinking attention with performers. arXiv preprint arXiv:2009.14794 (2020)
15. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 (2018)
16. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.-M.: Learning heuristics for the TSP by policy gradient. In: van Hoeve, W.-J. (ed.) CPAIOR 2018. LNCS, vol. 10848, pp. 170–181. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93031-2_12
17. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
18. Woo, H., Lee, H., Cho, S.: An efficient combinatorial optimization model using learning-to-rank distillation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 8, pp. 8666–8674 (2022)
19. Hua, W., et al.: Transformer quality in linear time. In: International Conference on Machine Learning, pp. 9099–9117. PMLR (2022)

20. Wu, C., et al.: Variance reduction for policy gradient with action-dependent factorized baselines. arXiv preprint arXiv:1803.07246 (2018)
21. Lovric, M., et al.: International Encyclopedia of Statistical Science. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-04898-2
22. Reinelt, G.: TSPLIB–a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
23. Applegate, D.: Concorde: a code for solving traveling salesman problems (2003). http://www.tsp.gatech.edu/concorde.html
24. Helsgaun, K.: An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Roskilde Roskilde Univ. **12**, 966–980 (2017)
25. Khalil, E., et al.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
26. Joshi, C.K., Laurent, T., Bresson, X.: On learning paradigms for the travelling salesman problem. arXiv preprint arXiv:1910.07210 (2019)
27. Luo, J., et al.: A graph convolutional encoder and multi-head attention decoder network for TSP via reinforcement learning. Eng. Appl. Artif. Intell. **112**, 104848 (2022)