Neural Networks 161 (2023) 589–597

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Memory-efficient Transformer-based network model for Traveling Salesman Problem

Hua Yang[a,*], Minghao Zhao[a], Lei Yuan[b], Yang Yu[b], Zhenhua Li[a], Ming Gu[a]

[a] *School of Software, Tsinghua University, Beijing, China*
[b] *National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China*

ARTICLE INFO

Article history:
Received 17 May 2022
Received in revised form 11 January 2023
Accepted 8 February 2023
Available online 16 February 2023

Keywords:
Combinatorial optimization
Deep reinforcement learning
TSP
Transformer

ABSTRACT

Combinatorial optimization problems such as Traveling Salesman Problem (TSP) have a wide range of real-world applications in transportation, logistics, manufacturing. It has always been a difficult problem to solve large-scale TSP problems quickly because of memory usage limitations. Recent research shows that the Transformer model is a promising approach. However, the Transformer has several severe problems that prevent it from quickly solving TSP combinatorial optimization problems, such as quadratic time complexity, especially quadratic space complexity, and the inherent limitations of the encoder and decoder itself. To address these issues, we developed a memory-efficient Transformer-based network model for TSP combinatorial optimization problems, termed Tspformer, with two distinctive characteristics: (1) a sampled scaled dot-product attention mechanism with $O(L\log(L))$ (L is the length of input sequences) time and space complexity, which is the most different between our work and other works. (2) due to the reduced space complexity, GPU/CPU memory usage is significantly reduced. Extensive experiments demonstrate that Tspformer significantly outperforms existing methods and provides a new solution to the TSP combinatorial optimization problems. Our Pytorch code will be publicly available on GitHub https://github.com/yhnju/tspFormer.

1. Introduction

Combinatorial optimization problem (Cook, Lovász, Seymour, et al., 1995; Hochba, 1997; Li et al., 2020; Papadimitriou & Steiglitz, 1998) has important applications across many domains, such as transportation, logistics, production planning, operations research (Arora, 1998; Helsgaun, 2017b; Optimization, 2018; Papadimitriou & Steiglitz, 1998; Vasek Chvatal, Applegate, Bixby, & Cook, 2006). The Traveling Salesman Problem (TSP) (Boese, 1995; Gutin & Punnen, 2006; Jünger, Reinelt, & Rinaldi, 1995) is a classic combinatorial optimization problem, and many combinatorial optimization problems can be reduced to the TSP. Traditional methods to solve combinatorial optimization problems include exact methods (Woeginger, 2003) and various heuristic approximation algorithms (Arora, 1998; Helsgaun, 2017b; Lin & Kernighan, 1973).

In recent years, the use of machine learning methods to solve combinatorial optimization problems has aroused a great deal of interest and great attention. In contrast, the machine learning method can have a faster solving speed, better generalization, and adaptability. Machine learning methods are likely to be suitable for many optimization tasks through heuristic methods of automatically discovering itself based on training data, so they need minor manual operations than all kinds of solvers (Google, 2018; Optimization, 2018; Vasek Chvatal et al., 2006) only optimize for one job. Compared with traditional methods, the learning methods can greatly reduce the time of test instances, which is the advantage of learning methods for solving combinatorial optimization problems such as the TSP. On large-scale problems, test instances can yield results in milliseconds using a trained model, while traditional methods take months.

Recently, the Transformer model (Vaswani et al., 2017) has shown superior performance in solving combinatorial optimization problems than Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) models. However, when the Transformer model is used to train large-scale TSPs, **insufficient computer memory becomes the most significant bottleneck**, and the training time is also very long. With the increase in the number of city nodes, the training time is getting longer and longer, and the memory requirement is getting larger and larger, which eventually leads to memory overflow. As shown in Fig. 1, the dynamic programming algorithm with $O(n^2 2^n)$ space complexity will terminate on the server with 32G memory due to out-of-memory when the number of city nodes reaches 27.

* Corresponding author.
E-mail address: yang-h17@mails.tsinghua.edu.cn (H. Yang).

https://doi.org/10.1016/j.neunet.2023.02.014
0893-6080/© 2023 Elsevier Ltd. All rights reserved.

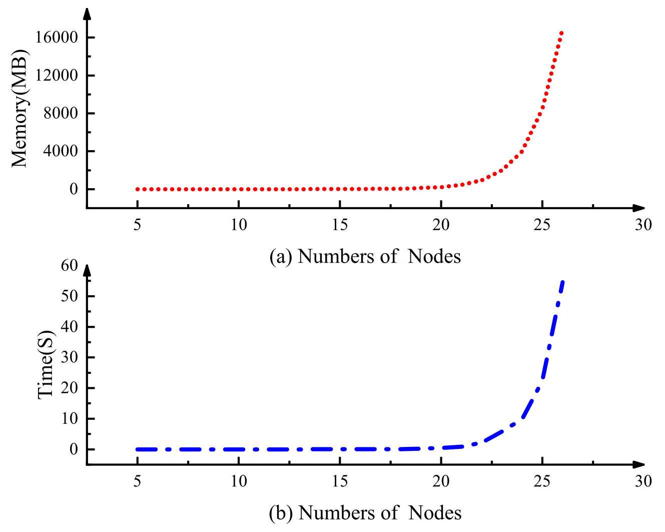

Fig. 1. Memory and time usage. (a) Memory usage, out-of-memory takes place if the number of nodes is more than 27. (b) Running time of the dynamic programming algorithm.

Vanilla Transformer model architecture (Vaswani et al., 2017), especially self-attention of Multi-Head Attention (MHA) Mechanism has three significant limitations when solving TSP combinatorial optimization problems:

- **Computational complexity is quadratic**. The operation of the scaled dot-product attention in the MHA causes $O(L^2)$ (L is the length of input sequences) of time and space complexity.
- **Memory requirements are too high resulting in out of memory**. When the number of city nodes is large, there will be a problem of insufficient CPU/GPU memory.
- Due to the limitations of the previous two, the scale of the problem to be solved is limited to a small scale, and **it is challenging to solve medium-scale and large-scale TSP problems**.

To solve these three problems, also to more effectively solve the TSP combinatorial optimization problem, we have thoroughly studied the self-attention mechanism of the Transformer model to meet the TSP-specific problem solutions. The contributions of this work are as follows:

- **Reduce time and space complexity of Transformer for the TSP combinatorial optimization field :** When using the Transformer to solve the TSP combinatorial optimization problem with an exhaustive search NP-hard problem of $O(L!)$ time complexity, we are the first to reduce the time and space complexity from $O(L^2)$ to $O(Llog(L))$, where L is the length of input sequences, i.e. the number of city nodes.
- **Reduce the memory usage of Transformer model:** By improving the MHA layer of the Encoder to reduce the space complexity and deleting the first layer of the decoder, the masked MHA layer, we have significantly reduced the number of parameters of the model, thereby reducing the memory usage.

In addition to this, the biggest difference between our work and existing work is in the following aspects:

- Compared with work (Bresson & Laurent, 2021), our work modified the transformer network model (Vaswani et al., 2017) to make the time complexity and space complexity of the model from $O(L^2)$ is reduced to $O(LlogL)$, which can increase the number of training nodes; (Bresson & Laurent, 2021)'s work is to train a model with the full structure of the transformer as much as possible, and then use this model to predict new sample instances.
- Compared with work (Kool, Van Hoof, & Welling, 2018), in addition to the above advantages, we did not train multiple baseline models such as critic or rollout, which improved the speed of training convergence; (Kool et al., 2018) is to solve all kinds of routing problems such as VRP, CVRP, etc. Kool et al. (2018) used graph embedding and the transformer to represent the features of the model.
- Compared with work (Zhou et al., 2021), we train the model with reinforcement learning on unlabeled data, and train and test with randomly generated data and real data, while the authors of (Zhou et al., 2021) train the model with supervised learning with labeled data, and they use their own datasets and public datasets for training and testing.

The above work and other work based on pointer network (Vinyals, Fortunato, & Jaitly, 2015), work based on graph neural network (Scarselli, Gori, Tsoi, Hagenbuchner, & Monfardini, 2008) and work based on local search (Hottung & Tierney, 2019) are all too complex in time and space. When the scale of city nodes becomes larger, there will be insufficient memory or the problem of too long training time. Therefore, this paper attempts to address these issues.

2. Related works

On the TSP, we take a quick look at the available machine learning-based methods and do not talk about the Non-learned methods for interested authors to take refer to Applegate et al. (2009), Helsgaun (2017a) and Rego, Gamboa, Glover, and Osterman (2011).

To solve TSP problems, in addition to exceptional solvers such as Concorde (Vasek Chvatal et al., 2006) Gurobi (Optimization, 2018), LKH3 (Helsgaun, 2017b), Google OR_Tools (Google, 2018), the use of neural networks has attracted more and more attention. Neural networks can learn better heuristic features from data to replace hand-crafted heuristic features in the TSP combinatorial optimization. Typical work using neural networks to solve combinatorial optimization problems includes the Hopfield-Nets (Hopfield & Tank, 1985) network which uses firstly neural networks to solve small-scale TSP problems, the Pointer Networks (Vinyals et al., 2015) which mainly use the attention mechanism (Bahdanau, Cho, & Bengio, 2014) to solve the variable length output problem, the Neural Combinatorial Optimization (Bello, Pham, Le, Norouzi, & Bengio, 2016) which uses Reinforcement Learning and RNN to solve TSP problems, the Vehicle Routing Problem (VRP) (Nazari, Oroojlooy, Snyder, & Takác, 2018) which solves the VRP problem using Reinforcement Learning.

Xing, Tu, and Xu (2020) solved the TSP using Monte Carlo tree search and deep reinforcement learning, which first converted the TSP into a tree search problem with a deep neural network. But it can only solve the small-scale TSP problem. Ma, Ge, He, Thaker, and Drori (2019) proposed a Graph Pointer Network to tackle the larger-scale TSP with time windows using hierarchical reinforcement learning and a graph embedding layer. But the accuracy of the result is not high. Barrett, Clements, Foerster, and Lvovsky (2020) proposed exploratory combinatorial optimization (ECO-DQN) to address any combinatorial optimization problem on a graph, and continuously improved the solution by learning to explore during the test phase not constructing the solution incrementally. More exploratory time is required during the testing phase. Joshi, Cappart, Rousseau, Laurent, and Bresson (2020) solved large-scale TSP through transfer learning and zero-shot generalization and trained on small-scale datasets for generalizing to large-scale unseen datasets. Sultana, Chan, Sarwar, and Qin (2022) proposed a non-Euclidean TSP network architecture that generalizes across all kinds of instances and can scale to instances that are larger than what was used during training time. Fu, Qiu, and Zha (2021) trained a small-scale model in supervised learning and generalized it to arbitrarily large size of the TSP instances using graph sampling and Monte Carlo tree search.

The work (Khalil, Dai, Zhang, Dilkina, & Song, 2017) learned heuristic algorithms with a combination of graph embedding and reinforcement learning (Sutton & Barto, 2018) on a diverse range of optimization problems over graphs such as the TSP, Minimum Vertex Cover, Maximum Cut. The work (Joshi, Laurent, & Bresson, 2019) used efficient TSP graph representation learning and the GCN (Graph Convolutional Networks) to output TSP sequence tours in a non-autoregressive manner, and improved the solutions of the tours with highly parallelized beam search.

In addition to the above work of reinforcement learning (Sutton & Barto, 2018) and graph neural network (Scarselli et al., 2008) solving the TSP and the VRP, the work of deep learning (Goodfellow, Bengio, Courville, & Bengio, 2016) to solve the TSP and the VRP has also performed quite well. The first work to solve the TSP with deep learning is the pointer network (Vinyals et al., 2015). The author of this paper uses LSTM to build a network model, and first uses the RNN deep learning model to solve combinatorial optimization problems such as the TSP. Then, the work of (Satyananda & Abdullah, 2021) reviews using deep learning to handle congestion in the VRP (Vehicle Routing Problems), which uses historical traffic data to train a deep learning network mode and inferred traffic prediction in future time. Furthermore, Xin, Song, Cao, and Zhang (2021) combine deep learning with the Lin–Kernighan–Helsgaun (LKH) for solving the TSP, which implements the combination of deep learning methods and traditional heuristic algorithms to solve the COP (combinatorial optimization problems). Sultana et al. (2022) use a deep learning approach to solve a variety of different TSP problems including a non-Euclidean TSP, a non-uniform distribution TSP. Also Kool, van Hoof, Gromicho, and Welling (2022) combine deep learning and dynamic programming algorithms to solve the TSP and the VRP.

More work is to combine deep learning and reinforcement learning (Sutton & Barto, 2018) to solve combinatorial optimization problems such as the TSP and the VRP. The first work of this type is Bello et al. (2016), which presents a framework to tackle combinatorial optimization problems such as the TSP using deep neural networks and reinforcement learning (Sutton & Barto, 2018). The work of Miki, Yamamoto, and Ebara (2018) learns the best solution instead of the optimal solution with deep learning and reinforcement learning for the TSP. The work of James, Yu, and Gu (2019) is to solve the online VRP with deep reinforcement learning, converting the online VRP to a VRP generation problem. The work of d O Costa, Rhuggenaath, Zhang, and Akcay (2020) learns a local search heuristic based on 2-opt operators by applying DRL (deep reinforcement learning). Ouyang, Wang, Han, Jin, and Weng (2021) generalize small instances to large ones in applying DRL (deep reinforcement learning) to solve the TSP. The work of García-Torres, Macias-Infante, Conant-Pablos, Ortiz-Bayliss, and Terashima-Marín (2022) proposes a memory-efficient algorithm to tackle the CVRP (Capacitated Vehicle Routing Problem) using deep learning and reinforcement learning approaches. Pan and Liu (2022) propose a novel framework to solve a dynamic VRP with deep learning and reinforcement learning. Chen, Huang, Zhang, and Wang (2022) use the DRL (deep reinforcement learning) method for the VRPTW (vehicle routing problem with time windows) and propose a novel end-to-end deep reinforcement learning method with a two-stage training strategy for the VRPTW.

The following three tasks are highly related to the work of this paper, and they all use Transformer to solve TSP problems. The first work of using the Transformer (Vaswani et al., 2017) to solve the TSP problems is Deudon, Cournut, Lacoste, Adulyasak, and Rousseau (2018). This paper used the standard Encoder of the Transformer without the usage of the positional encoding, and used the recently visited three city nodes as the characteristic input of the Decoder. At the same time, it used the 2-opt heuristic algorithm (Johnson, 1990) to enhance the accuracy of the results predicted by the Transformer model.

However, the work that has an important impact is Kool et al. (2018). This paper used the Transformer model to solve path optimization problems, including the TSP, the Vehicle Routing Problem (VRP), the Orienteering Problem (OP), etc. It used graph embedding to represent the features of the location coordinates of city nodes, and the encoder is also an Encoder of a standard Transformer without position encoding, and used the first city node visited and the last seen city node as an input to the Decoder. The latest work in this area is Bresson and Laurent (2021). This paper keeps the complete Transformer structure as much as possible to solve the TSP problems. Unlike the standard Transformer, the Encoder does not use positional encoding. The Decoder uses all partially visited city nodes as the input of the Decoder.

Obviously, the biggest difference between the work of this paper and the work of these three papers is the time and space complexity is reduced to $O(L\log(L))$, where L is the length of input sequences, i.e. the number of city nodes. The time and space complexity of the Transformer network structure in the above three papers are all $O(L^2)$, and the Decoders are all autoregressive methods, one node at a time, which are all constructive solutions, and the number of solved nodes did not exceed 100.

Part of the above work uses supervised learning to design network models, but these supervised learning-based approaches need a huge number of pre-computed TSP solutions, making them challenging to apply to large-scale situations. Therefore, the majority of the above work used reinforcement learning to model the network architecture and train the network model. In addition to the regular TSP problems, there are various variants of TSP such as the decision TSP (Prates, Avelar, Lemos, Lamb, & Vardi, 2019), the multiple TSP (Kaempfer & Wolf, 2018), the VRP (vehicle routing problem) (Kool et al., 2018; Lu, Zhang, & Yang, 2019; Nazari et al., 2018).

3. Preliminaries

3.1. Problem formulation

In this work, we focus only on solving the two-dimensional Euclidean plane-symmetric TSP, which is a complete and undirected graph. Given a problem instance of the TSP graph s, the node feature is represented as the node location coordinate x_i, $i \in \{1, 2, 3, \ldots, n\}$. We are concerned with finding an optimal route permutation of the nodes π such that each city is visited only once and has the minimum total distance $L(\pi|s)$. The objective function of the TSP can be formalized as the following constrained optimization problem:

$$\begin{aligned} \min_{\pi} \quad & L(\pi|s) = \sum_{i=1}^{n-1} \|x_{\pi_i} - x_{\pi_{i+1}}\|_2 + \|x_{\pi_1} - x_{\pi_n}\|_2 \\ s.t. \quad & f(\pi, s) = 0, \\ & g(\pi, s) \leq 0. \end{aligned} \tag{1}$$

where $f(\pi, s)$ and $g(\pi, s)$ is constraint functions.

We learn the solutions of the problem instances using Reinforcement Learning (Sutton & Barto, 2018). Therefore, we define

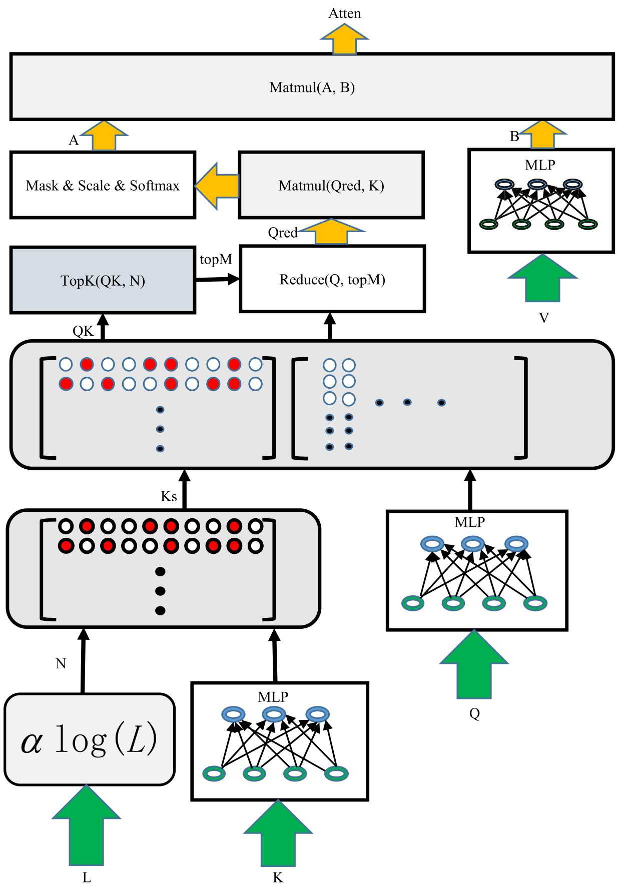

Fig. 2. The Tspformer architecture. The green arrows represent the input, and the small red dots are randomly sampled data points, and the final output is Atten.

rewards, *actions*, *transition*, *states*, and *policy*, which are essential variables in the Markov Decision Process of Reinforcement Learning, as follows:

- States: The state s_t represents an ordered sequence of visited city nodes in the TSP instances. The initial state s_0 is NULL, the first state s_1 is the index order of the first visited city, and the final state s_n is the collection of all the visited city nodes.
- Actions: The action a_t is one of the unvisited city nodes, in other words, the next node which will be selected.
- Rewards: The reward value is defined as the negative distance cost, i.e. $r(a_i, s_i) = -\|x_{\pi_i} - x_{\pi_{i+1}}\|_2$.
- Transition: The state transition of the traveling salesman problem is to select one from the collection of city nodes that have never been visited and add it to the collection of city nodes that have been seen.
- Policy: The policy $p_\theta(\pi|s)$ is represented as a neural network and the parameter θ is the trainable weights of the network. Given a set of visited city nodes, this policy function $p_\theta(\pi|s)$ will return the probability distribution of the next unselected candidate city. This policy function $p_\theta(\pi|s)$ is defined as Eq. (2) using a probability chain rule.

$$p_\theta(\pi|s) = \prod_{i=1}^{n} p_\theta(\pi_i|\pi_{1\ldots i-1}, s) \tag{2}$$

4. The model architecture

Our proposed Tspformer holds the architecture of the encoder and decoder while addressing the TSP combinatorial optimization problem. The model architecture of Tspformer is illustrated in Fig. 2, coming from the overall structure of the vanilla Transformer (Vaswani et al., 2017) but making the following essential changes:

1. **Reduce time and space complexity:** We sample the query data of the decoder so that the time and space complexity of the score function of the attention mechanism, i.e., scaled dot-product attention function (Vaswani et al., 2017; Zhou et al., 2021), is reduced from $O(L^2)$ to $O(Llog(L))$, where L is the length of input sequences, i.e. the number of city nodes. Please refer to Fig. 2, and we will discuss in detail in the next section.
2. **Delete the first module of the Decoder** in the Transformer architecture, i.e., Masked Multi-Head Attention module.
3. **Remove the positional encoding of the Encoder** in the Transformer architecture, i.e., Positional Encoding module.

Concerning the model, the main novelty between the Informer (Zhou et al., 2021) model as presented in Zhou et al. (2021) and our presented Tspformer model is as follows:

(1) in an encoder module, we do not use a self-attention distilling operation, while the Informer (Zhou et al., 2021) model utilizes it indeed.

(2) in the scaled dot-product attention module, we shuffle the input data and sample randomly from them, while the Informer (Zhou et al., 2021) model emphasizes the order of the input.

(3) in the input module, we use $\alpha log(L)$ as the length of the input, while the Informer (Zhou et al., 2021) model does not.

(4) in the decoder module, we delete the first module of the decoder, scaled dot-product attention module, while the Informer (Zhou et al., 2021) uses it.

(5) in the output of the decoder module, we autoregressive output one node at a time, while the Informer (Zhou et al., 2021) model is one shot at non-autoregressive and includes partial zeros in the whole solution.

4.1. Tspformer architecture

The Multi-Head Attention (MHA) module is the main component of the Transformer (Vaswani et al., 2017), and the Scaled Dot-Product Attention in MHA is the most time-consuming calculation. In terms of the query sparsity from Informer (Zhou et al., 2021), the score function of Scaled Dot-Product Attention forms an extended tail distribution, i.e., the primary attention comes from a few dot-product values, and others can be negligible. Therefore, the query data can be sampled, resulting in fewer data joining the matrix multiplication operation.

As shown in Fig. 2, the input of the Tspformer architecture is a query, key, value, and the length of the input data sequence, denoted as Q, K, V, L, respectively. The MHA of the Transformer is defined as

$$\begin{aligned} &MultiHeadAttention(Q, K, V) \\ &= Concat(Head_1, Head_2, \ldots, Head_h)W^o \end{aligned} \tag{3}$$

where $Q, K, V \in \mathbb{R}^{H\times d_m}$ are the input matrices, H is the sequence length, h is the number of the head, and d_m is the embedding dimension.

The sample number of the key is $N = \alpha \log(L)$, α is fine-tuning parameters, usually set to $1 \sim 10$, i.e., $\alpha \in [1, 10]$. The key is sampled representing as $K_{sample} = Sample(K, N)$. The compatibility of the query and key is computed using the following matrix multiplication, i.e. Eq. (4), and its result Q_E takes the first N values with the highest probability which index is $TopM$, and then in terms of the $TopM$ the data query is sampled, i.e., Q_{red}. Again, we computed the compatibility of the query and key with Eq. (5). After masking the visited city nodes, we can compute the attention value such as Eq. (6). The overall process of efficient Sampled Scaled Dot-Product Attention Mechanism shows in Algorithm 1, and the proof of its mathematical theory refers to Informer (Zhou et al., 2021).

$$Q\_E = MatMul(W^{Q}Q, W^{K_{sample}}K_{sample}) \tag{4}$$

$$Q\_Reduce = MatMul(W^{Q_{red}}Q_{red}, W^{K}K) \tag{5}$$

$$\begin{aligned} Head_i &= AttSampled(Q_{red}W_i^{Q_{red}}, KW_i^{K}, VW_i^{V}) \\ &= softmax(\frac{Q_{red}W_i^{Q_{red}}(KW_i^{K})^T}{\sqrt{d_k}})VW_i^{V} \end{aligned} \tag{6}$$

where $W_i^{Q_{red}}, W_i^{K} \in \mathbb{R}^{d_m \times d_k}$, $W_i^{V} \in \mathbb{R}^{d_m \times d_v}$ are the learnable matrices, and d_k, d_v is the hidden dimensions of the linear projection, and we suppose $d_k = d_m = d_v = d$.

Encoder It is a standard Encoder of Transformer with residual connection and batch normalization replacing for layer normalization, except for scaled dot-product attention replacing for sampled scaled dot-product attention.

Decoder The Decoder of Tspformer removed the first component of the standard Transformer, i.e., Masked Multi-Head Attention module. The others are the same as Transformer model. It uses autoregressive step by step, one city node at a time, and the Greedy search and Beam search to refine the solution space.

Algorithm 1 Sampled Scaled Dot-Product Algorithm

```
Input: $Q, K, V \in \mathbb{R}^{L \times d_m}, L \in \mathbb{R}^n$
Output: atten
1: Initialize $\alpha \in [1, 10], N = \alpha \log(L)$
2: randomly select N data from K as $K_{sample}$
3: compute score function $Q\_E = QK_{sample}$
4: compute M = max(Q_E) - mean(Q_E)
5: set TopM from M as $Q_{red}$
6: mask $Q_{red}$ = $Q_{red} \bigodot M_{mask}$
7: compute score value atten = softmax($\frac{Q_{red}K^T}{\sqrt{d}}$)V
8: return atten
```

4.2. Model training with reinforcement learning

In terms of combinatorial optimization, Reinforcement Learning (Sutton & Barto, 2018) may give an acceptable approach for training neural networks. To reduce the parameters of the Tspformer network, we employ model-free policy-based Reinforcement Learning (Sutton & Barto, 2018) to train the network model. The loss function is the ATL (Average Tour Length) which is defined as Eq. (7), given an input graph s

$$C(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}L(\pi|s) \tag{7}$$

We use stochastic gradient descent and policy gradient methods to optimize the Tspformer parameters. The well-known REINFORCE algorithm (Williams, 1992) is used to compute the update of the gradient, i.e., Eq. (8).

$$\nabla_\theta C(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(L(\pi|s) - b(s))\nabla_\theta \log p_\theta(\pi|s)] \tag{8}$$

where $b(s)$ is a baseline independent of π and predicts the ATL to decrease gradient variance. A good $b(s)$ baseline decreases gradient variance and speeds up learning. We select the Greedy algorithms of decoding as the baseline.

In order to construct mini-batch B and sample a single tour, we sample graphs $s_1, s_2, \ldots, s_B \sim S$, with Monte Carlo sampling, the gradient in Eq. (8) is approximated as follows:

$$\nabla_\theta C(\theta) = \frac{1}{B}\sum_{i=1}^{B}(L(\pi_i|s_i) - b(s_i))\nabla_\theta \log p_\theta(\pi_i|s_i) \tag{9}$$

5. Experiments

5.1. Datasets and experimental details

We use datasets generated artificially, which are randomly generated in the unit square [0,1]*[0,1] from a uniform distribution, and the real-world public dataset, named TSPLIB (Heidelberg, 2005). The test instances and the training instances must be independent and identically distributed (i.i.d), and we use a uniform distribution to sample between 0 and 1. Other forms of data need to be normalized first to move between 0 and 1.

All models are trained/tested on a single piece of Geforce RTX 2080Ti GPU with 11G GPU memory and CPU E5 2678 v3 with 32G CPU memory. Pytorch1.8 is used in the experiments. When the number of city nodes does not exceed 200, we set the mini-batches to 32, when the number of nodes is 400 and 600, set it to 16, and when the number of nodes is 800, set it to 5, and the number of nodes is 1000, set it to 4.

Tspformer has an Encoder of 6 stacks with $h = 8$ heads and $d = 256$ hidden dimensions, and a Decoder of 2 stacks with only cross-attention component and feed-forward network. Our proposed approaches are optimized utilizing the Adam optimizer, whose learning rate starts at $1e^{-4}$ and decays 10 times per 2 epochs, for a total of 10 epochs.

5.2. Results and analysis

The comparison between our proposed Tspformer and other similar works shows in Table 1. To evaluate the effectiveness of our model and other similar baseline works, we present two types of metrics: Average Tour Length (ATL), and Optimality Gap Ratio (OGR), which is the average percentage ratio of the anticipated tour length compared to optimal solutions. The optimal solution obtained by the three solvers Concorde, Gurobi, and OR_Tools is a comparison baseline for all other tasks.

From Table 1, we observe that:

(1) the Tspformer can find the ATL of 1000 city nodes, while other works can only find that of no more than 200 nodes because of out-of-memory.

(2) from the data of 20, 100, and 200 nodes, it can be seen that the performance of Tspformer is slightly reduced, but the memory usage and training time are reduced. Compared with the general neural network in the middle row and the transformer-based model structure in the "Tspformer" row, the OGR value of Tspformer is slightly larger. However, from Tables 3 and 4, we can find that the training time and the memory usage is significantly reduced.

(3) As can be seen from the last 2 rows of Table 1, after using Greedy search and Beam search, the performance of all kinds of the TSP instances is improved highly.

(4) using neural network structures such as CNN, RNN, and transformer can achieve near-optimal performance, such as Bello et al. OGR 1.04%, ATL 3.87, and Kook et al. OGR 0.26%, ATL 3.84, is close to the optimal solution of 3.83 in the TSP20.

(5) we use 500, 750, and 1000 nodes to train a model directly, and then use these trained models to test the data while we do not use the model trained with 200 nodes, and then generalize to 500, 750, and 1000 nodes to get test data. The important significance of this is that a model trained with 1000 nodes can

Table 1
Comparison of Tspformer with three solvers and similar work by other authors. All results are averaged based on 100 random instances. The number of city nodes in the TSP graph are 20, 100, 200, 500, 750 and 1000. ATL, i.e., Average Travel Length, and OGR (Optimal Gap Ratio). GS (Greedy Search), BS (Beam Search with the width of search 10). The '-' indicates out-of-memory.

Method	TSP20		TSP100		TSP200		TSP500		TSP750		TSP1000	
	ATL	OGR	ATL	OGR	ATL	OGR	ATL	OGR	ATL	OGR	ATL	OGR
Concorde	3.83	0.00%	7.77	0.00%	10.53	0.00%	16.58	0.00%	20.14	0.00%	23.12	0.00%
Gurobi	3.83	0.00%	7.77	0.00%	10.53	0.00%	16.57	0.00%	–	–	–	–
OR-Tools	3.84	0.26%	7.97	2.57%	11.78	11.87%	17.46	5.3%	22.41	11.3%	26.48	14.5%
Bello et al.	3.87	1.04%	8.33	7.21%	13.43	27.54%	–	–	–	–	–	–
Dai et al.	3.89	1.57%	8.28	6.56%	13.87	31.72%	–	–	–	–	–	–
Nazari et al.	3.97	3.66%	8.44	8.62%	13.02	23.65%	–	–	–	–	–	–
Joshi et al.	3.83	0.00%	8.05	3.60%	12.94	22.89%	–	–	–	–	–	–
Deudon et al.	3.84	0.26%	8.85	13.90%	12.64	20.04%	–	–	–	–	–	–
Kool et al.	3.84	0.26%	7.94	2.19%	13.22	25.54%	–	–	–	–	–	–
Bresson et al.	3.89	1.57%	7.79	0.26%	12.37	17.47%	–	–	–	–	–	–
Tspformer.	3.91	2.09%	8.39	7.98%	13.53	28.49%	**18.27**	**10.19%**	**23.79**	**18.12%**	**29.89**	**29.28%**
Tspformer(GS).	3.89	1.57%	7.96	2.45%	13.26	25.93%	**18.02**	**8.69%**	**23.06**	**14.50%**	**28.33**	**22.53%**
Tspformer(BS).	3.84	0.26%	7.79	0.26%	13.03	23.73%	**17.57**	**5.97%**	**22.89**	**13.65%**	**27.02**	**16.87%**

Table 2
The training time and testing time comparison of Tspformer with similar work by other authors. We take 100 randomly generated test data and take the sum of their total test time, the unit of time is seconds. The number of city nodes in the TSP graph are 20, 100, 200, 500, 750 and 1000. GS (Greedy Search), BS (Beam Search with the width of search 10). Similar work by other authors also uses a beam search method with a width of 10, which is exactly the same as the parameters used by our beam search method Tspformer(BS). The '-' indicates out-of-memory.

method	TSP20		TSP100		TSP200		TSP500		TSP750		TSP1000	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
Bello et al.	2 h 29 m	1.8 s	37 h 12 m	78.3 s	67 h 49 m	84.6 s	–	–	–	–	–	–
Dai et al.	2 h 51 m	2.5 s	48 h 29 m	61.3 s	72 h 32 m	85.2 s	–	–	–	–	–	–
Nazari et al.	2 h 42 m	2.1 s	45 h 48 m	60.5 s	81 h 10 m	91.5 s	–	–	–	–	–	–
Joshi et al.	2 h 37 m	1.9 s	39 h 27 m	59.8 s	74 h 19 m	82.4 s	–	–	–	–	–	–
Deudon et al.	2 h 40 m	3.2 s	38 h 31 m	57.2 s	69 h 73 m	97.5 s	–	–	–	–	–	–
Kool et al.	2 h 41 m	2.5 s	39 h 43 m	72.4 s	71 h 39 m	98.9 s	–	–	–	–	–	–
Bresson et al.	2 h 52 m	2.4 s	38 h 33 m	69.6 s	66 h 27 m	96.2 s	–	–	–	–	–	–
Tspformer(GS).	2 h 12 m	1.1 s	35 h 43 m	3.5 s	61 h 39 m	6.4 s	**81 h 28 m**	**7.3 s**	**103 h 38 m**	**9.5 s**	**124 h 26 m**	**14.6 s**
Tspformer(BS).	2 h 12 m	1.6 s	35 h 43 m	54.2 s	61 h 39 m	77.3 s	**81 h 28 m**	**143.7 s**	**103 h 38 m**	**182.4 s**	**124 h 26 m**	**235.8 s**

Table 3
Tspformer training time per epoch with 100 iterations. The time unit of each data is seconds. The '-' indicates out-of-memory.

Works	TSP20	TSP50	TSP100	TSP200	TSP400	TSP600	TSP800	TSP1000
Deudon et al.	21.16	46.52	88.23	180.35	–	–	–	–
Kool et al.	19.25	43.11	81.79	178.42	–	–	–	–
Bresson et al.	20.43	45.76	82.57	179.38	–	–	–	–
Tspformer.	**15.89**	**32.82**	**62.17**	**125.68**	**254.26**	**392.29**	**503.24**	**672.57**

Table 4
Comparison for Tspformer memory usage. The unit of each data is MB. The '-' indicates out-of-memory.

Works	TSP20	TSP50	TSP100	TSP200	TSP400	TSP600	TSP800	TSP1000
Deudon et al.	1347	1591	2312	5141	–	–	–	–
Kool et al.	1301	1498	2283	5101	–	–	–	–
Bresson et al.	1321	1587	2299	5125	–	–	–	–
Tspformer.	**1287**	**1467**	**1917**	**3713**	**5893**	**6343**	**6809**	**7967**

be used to infer the test data of up to 10 000 nodes, which shows in Table 5.

The training time and testing time comparison of Tspformer with similar work by other authors show in Table 2. For a fair comparison, similar work by other authors also uses a beam search method with a width of 10, which is exactly the same as the parameters used by our beam search method Tspformer (BS). The training model used by Tspformer (GS) and Tspformer (BS) is the same, but the search method used in the inference phase is different, so the training time of the model is the same. For the same training model, due to the different search methods used, the inference time used is also very different. The inference time used by the greedy method is very small, while the beam search method requires a lot of inference time. However, the accuracy of the beam search method is much higher than that of the greedy method. The beam search method uses time in exchange for the accuracy of the solution. From the examples of TSP20, TSP100, and TSP200, it can be seen that Tspformer (BS) uses less time than other similar works, whether it is the training time of the model or the inference time of the inference test phase. From the examples TSP500, TSP750, and TSP1000, it can be seen that when the number of nodes exceeds 200, the similar work of other authors cannot train the model due to insufficient memory. Only our method Tspformer can train such a large node model.

From Table 3, we see that the training time of Tspformer is less than that of other similar jobs. We only compared three very similar works which use the Transformer network structure to solve the TSP problems. The training time of Tspformer per epoch with 100 iterations is reduced by about 30.2%(54 s), 29.8%(53 s), 30.6%(55 s)(Bresson et al. 179.38 s, Kool et al. 178.42 s, Deudon et al. 180.35 s and Tspformer 125.68 s) in TSP200. When the number of nodes exceeds 200, other similar tasks cannot run due

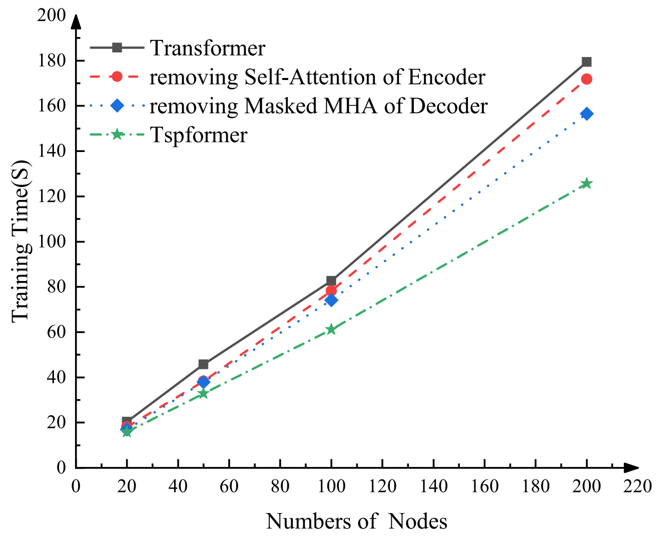

Fig. 3. Ablation of training time.

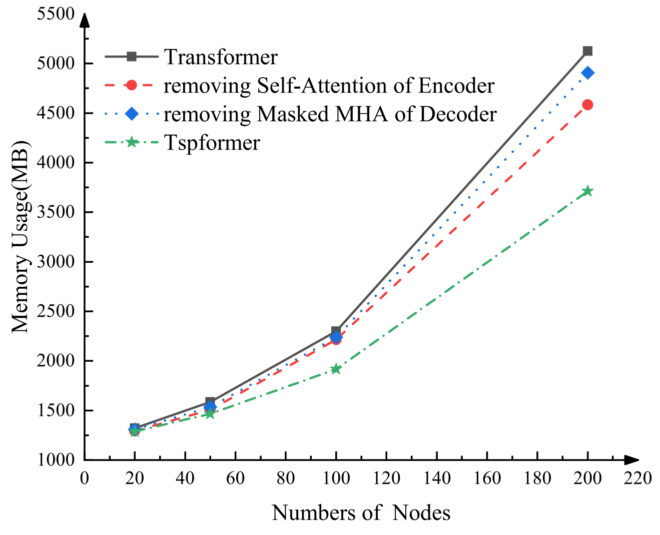

Fig. 4. Ablation of memory usage.

to insufficient memory. However, Tspformer can continue to run on the scale of 400, 600, 800, and 1000 city nodes. But the running time will be greatly increased, such as 672.57 s per epoch for TSP1000 task.

Table 4 is a comparison for memory usage of Tspformer with other similar works. As in Table 4, when the number of city nodes reaches 800 and 1000, the similar work of other authors has to stop running due to insufficient memory. Only Tspformer can run normally on relatively large-scale nodes such as 800 and 1000. For example, TSP1000 requires 7967 MB of memory. The memory usage of Tspformer is reduced by about 1400 MB than that of Bresson et al. Kool et al. and Deudon et al.

Table 5 shows a performance comparison of training on small TSPs and testing on large TSPs. We compare with kool's work (Kool et al., 2018), the data in brackets is the test data of kool's work. When the number of nodes exceeds 200, the kool's work cannot train the model due to insufficient memory. When the number of nodes is 20, 100, and 200, our work is compared with that of kool. The data in the table is the optimal ratio. The optimal ratio is the ratio of the travel length predicted by the model to the optimal value. The optimal value is obtained by Concorde and Gurobi. Compared with kool's work, our work has the following advantages:

(1) We can train models on a scale of more than 200 nodes, while kool cannot;

(2) On small-scale problems, except for a few, the optimal ratio of most TSP instances is smaller than that of kool;

(3) We found that a model trained on a small scale can generalize to a 10X or even 100X larger model;

(4) We can also find that the model trained on 1000 nodes can also test the data of 20 nodes, that is, the model trained on a larger model can also be tested on a smaller scale, reflecting good generalization.

5.3. *Refining accuracy of results with greedy search and beam search*

We use greedy search and beam search algorithms to improve the accuracy of model prediction results, and the results are shown in Table 1. After using greedy search and beam search, Average Travel Length (ATL) and Optimal Gap Ratio (OGR) are all reduced in all TSP instances. In the TSP20 instance, the ATL reduced from 3.91 to 3.89 and the OGR reduced from 2.09% to 1.57% for Greedy search, the ATL from 3.89 to 3.84 and the OGR from 1.57% to 0.26% for Beam search with the search width 10. Similarly, the ATL reduced from 18.27 to 18.02 and the OGR reduced from 10.19% to 8.69% for Greedy search, the ATL from 18.02 to 17.57 and the OGR from 8.69% to 5.97% for Beam search with the search width 10 in the TSP500 instance.

It can be seen that through greedy search and beam search technology, the accuracy of the prediction results is greatly improved, and it can compete with the work of other authors.

5.4. *The ablation*

We also conducted additional ablation experiments comparing the training time and memory usage.

As shown in Fig. 3, we can see that the training time of Tspformer is less than removing the masked Multi-Head Attention module of the Decoder in Transformer, removing the Multi-Head Attention module of the Encoder in Transformer, and the overall Transformer architecture. As the number of city nodes increases, more training time is saved in Tspformer network architecture.

Fig. 4 compares the ablation experiment of memory usage. If only the masked Multi-Head Attention module of the Decoder is deleted, the memory usage of the Transformer is slightly reduced. If only the Multi-Head Attention module of the Encoder is deleted, the memory usage of the Transformer will be more reduced. This is because the Encoder is set to 6 layers, and more layers occupy more memory space. However, when we use Tspformer, its memory usage will be greatly reduced. As the number of city nodes increases, more memory usage is saved in Tspformer network architecture.

5.5. *Experiments on real-world datasets*

To evaluate our approach Tspformer, we performed experiments on publicly available benchmark real-world dataset TSPLIB (Heidelberg, 2005) which includes 110 problems in total in addition to the synthetic data experiments. As shown in Table 6, we select only 12 of 110 problems in total to evaluate our method. We randomly selected 12 of 110 problems from TSPLib for testing according to the number of nodes from small to large. Due to most of the data are similar, so we selected these 12 data representatively, and the node sizes are about 50, 100, 150, 200, 300. We use randomly generated data with node sizes of 50, 100, 150, 200, and 300 to train their corresponding models, and then we use their respective models to test these 12 data that are close to

Table 5
Performance comparison of training on small TSPs and testing on large TSPs. We compare with the work of kool (Kool et al., 2018), the data in brackets is the test data of his work. The data in the table is the optimal ratio. The optimal ratio is the ratio of the travel length predicted by the model to the optimal value. The optimal value is obtained by Concorde and Gurobi.

Test								
Train	TSP20	TSP100	TSP200	TSP500	TSP750	TSP1000	TSP10 000	TSP100 000
TSP20	1.008(1.007)	1.017(1.019)	1.122(1.186)	1.141(1.182)	1.158(1.167)	1.233(1.247)	1.364(1.397)	1.823(1.912)
TSP100	1.083(1.089)	1.015(1.018)	1.104(1.112)	1.132(1.126)	1.145(1.152)	1.183(1.181)	1.252(1.236)	1.801(1.859)
TSP200	1.144(1.142)	1.137(1.143)	1.026(1.047)	1.129(1.137)	1.138(1.203)	1.161(1.163)	1.221(1.218)	1.792(1.784)
TSP500	1.212	1.153	1.125	1.037	1.114	1.153	1.192	1.753
TSP750	1.332	1.225	1.192	1.135	1.024	1.104	1.167	1.687
TSP1000	1.357	1.312	1.237	1.215	1.125	1.038	1.115	1.526

Table 6
The Tour Length (TourL) of the real-world dataset TSPLIB. OGR (Optimal Gap Ratio).

Instance	OPT	Deudon et al.		Kool et al.		Bresson et al.		Tspformer.	
		TourL	OGR	TourL	OGR	TourL	OGR	TourL	OGR
eil51	426	441	3.52%	431	1.17%	439	3.05%	437	2.58%
st70	675	691	2.37%	687	1.78%	694	2.81%	703	4.15%
kroA100	21 282	22 396	5.76%	22 375	4.92%	22 384	5.17%	22 391	5.21%
rd100	7910	7962	0.66%	7951	0.52%	7986	0.96%	8053	1.81%
lin105	14 379	14 673	2.03%	14 602	1.56%	14 692	2.18%	14 706	2.32%
pr124	59 030	63 496	7.83%	62 102	5.21%	63 471	7.52%	63 709	7.93%
ch130	6110	6224	1.82%	6195	1.34%	6215	1.71%	6289	2.93%
ch150	6528	6698	2.53%	6682	2.42%	6691	2.50%	6721	2.96%
u159	42 080	43 293	2.88%	43 195	2.65%	43 218	2.72%	43 437	3.22%
kroA200	29 368	31 505	7.27%	31 478	7.18%	31 611	8.12%	31 887	8.58%
tsp225	126 643	139 278	9.98%	138 413	9.29%	139 002	9.64%	139 704	10.31%
a280	2579	2671	3.57%	2606	1.05%	2638	1.54%	2696	4.54%

the number of nodes of our own model. The corresponding data are 50 for eil51, st70; 100 for kroA100, rd100, lin105, pr124; 150 for ch130, ch150, u159; 200 for kroA200, tsp225; 300 for a280 respectively. We convert the 12 data selected from the TSPLib dataset to the interval [0, 1]. The conversion rule is that the data is 1-digit or 2-digit divided by 100, 3-digit divided by 1000, so analogy. The final predicted result is multiplied by 100, 1000, and so on. Because these actual data are only a special case of randomly generated data, they also satisfy the requirements of i.i.d. In terms of model selection, we choose the model closest to the integer. For example, eil51 selects a randomly generated training model of 50 nodes for testing, kroA100 and lin105 select a randomly generated training model of 100 nodes for testing, and other data are similar.

Compared with similar work by other authors, Tspformer's OGR (Optimal Gap Ratio) is the largest, TourL (Tour Length) is the most considerable travel distance. Still, we traded less training time and memory usage at the cost of less performance loss.

In Table 4, the optimal gap is not the best, because this is not what we want to solve, the problem we want to solve is the scale of the problem rather than the accuracy of the problem solution. Many authors have done good work on the accuracy of the problem, but the solution of large-scale problems has been unsolvable for a long time due to memory constraints, which is the problem that this paper aims to solve. By sampling, the scale of the problem is enlarged at the expense of the accuracy of the solution. Of course, if we use greedy search and beam search as in Table 1, the accuracy of each test will be greatly improved.

More importantly, instead of generalizing a model trained with 100 nodes to a test case of 1000 nodes, we use the model trained with 1000 nodes to directly solve the test case of 1000 nodes. The model trained from these 1000 nodes can generalize to 10 000 nodes or even 100 000 node test cases, which shows in Table 5. By analogy, we can use the model trained with 10,000 nodes to generalize to the test case of 1,000,000 nodes, so that we can quickly obtain approximate solutions to large-scale problems. This has important applications in the chip design of integrated circuits and so on.

6. Conclusion

We studied combinatorial optimization such as the TSP problems and proposed Tspformer to solve this problem in this paper. Specifically, we designed the sampled scaled dot-product attention mechanism to address the challenges of quadratic memory utilization and computational complexity in vanilla Transformer. Also, the deleting of the Masked Multi-Head Attention module of the decoder alleviates the constraint of standard encoder–decoder architecture in terms of TSP combinatorial optimization. Experiments on real-world data show the effectiveness of Tspformer in solving TSP problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I gave a link in my paper.

Acknowledgment

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W., Espinoza, D. G., Goycoolea, M., et al. (2009). Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, *37*(1), 11–15.
Arora, S. (1998). The approximability of NP-hard problems. In *Proceedings of the thirtieth annual ACM symposium on theory of computing* (pp. 337–348).
Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
Barrett, T., Clements, W., Foerster, J., & Lvovsky, A. (2020). Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 34* (pp. 3243–3250).

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940.

Boese, K. D. (1995). *Cost versus distance in the traveling salesman problem*. Citeseer.

Bresson, X., & Laurent, T. (2021). The transformer network for the traveling salesman problem. arXiv preprint arXiv:2103.03012.

Chen, J., Huang, H., Zhang, Z., & Wang, J. (2022). Deep reinforcement learning with two-stage training strategy for practical electric vehicle routing problem with time windows. In *International conference on parallel problem solving from nature* (pp. 356–370). Springer.

Cook, W., Lovász, L., Seymour, P. D., et al. (1995). *Combinatorial optimization: papers from the DIMACS special year, Vol. 20*. American Mathematical Soc..

d O Costa, P. R., Rhuggenaath, J., Zhang, Y., & Akcay, A. (2020). Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian conference on machine learning* (pp. 465–480). PMLR.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., & Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research* (pp. 170–181). Springer.

Fu, Z.-H., Qiu, K.-B., & Zha, H. (2021). Generalize a small pre-trained model to arbitrarily large TSP instances. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 35* (pp. 7474–7482).

García-Torres, R., Macias-Infante, A. A., Conant-Pablos, S. E., Ortiz-Bayliss, J. C., & Terashima-Marín, H. (2022). Combining constructive and perturbative deep learning algorithms for the capacitated vehicle routing problem. arXiv preprint arXiv:2211.13922.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning, Vol. 1*. MIT press Cambridge.

Google, I. (2018). Google optimization tools(or-tools). URL: https://github.com/google/or-tools.

Gutin, G., & Punnen, A. P. (2006). *The traveling salesman problem and its variations, Vol. 12*. Springer Science & Business Media.

Heidelberg, U. (2005). TSPLIP datasets. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html.

Helsgaun, K. (2017a). *An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems* (pp. 24–50). Roskilde: Roskilde University.

Helsgaun, K. (2017b). *An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems: Technical report*: *Technical report*.

Hochba, D. S. (1997). Approximation algorithms for NP-hard problems. *ACM Sigact News*, *28*(2), 40–52.

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, *52*(3), 141–152.

Hottung, A., & Tierney, K. (2019). Neural large neighborhood search for the capacitated vehicle routing problem. arXiv preprint arXiv:1911.09539.

James, J., Yu, W., & Gu, J. (2019). Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *20*(10), 3806–3817.

Johnson, D. (1990). Local search and the traveling salesman problem. In *Lecture notes in computer science, Proceedings of 17th international colloquium on automata languages and programming* (pp. 443–460). Berlin: Springer-Verlag, 1990.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., Laurent, T., & Bresson, X. (2020). Learning TSP requires rethinking generalization. arXiv preprint arXiv:2006.07054.

Joshi, C. K., Laurent, T., & Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem. arXiv preprint arXiv:1906.01227.

Jünger, M., Reinelt, G., & Rinaldi, G. (1995). The traveling salesman problem. *Handbooks in Operations Research and Management Science*, *7*, 225–330.

Kaempfer, Y., & Wolf, L. (2018). Learning the multiple traveling salesmen problem with permutation invariant pooling networks. arXiv preprint arXiv:1803.09621.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in neural information processing systems* (pp. 6348–6358).

Kool, W., van Hoof, H., Gromicho, J., & Welling, M. (2022). Deep policy dynamic programming for vehicle routing problems. In *International conference on integration of constraint programming, artificial intelligence, and operations research* (pp. 190–213). Springer.

Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems!. arXiv preprint arXiv:1803.08475.

Li, W., Ding, Y., Yang, Y., Sherratt, R. S., Park, J. H., & Wang, J. (2020). Parameterized algorithms of fundamental NP-hard problems: a survey. *Human-Centric Computing and Information Sciences*, *10*(1), 1–24.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, *21*(2), 498–516.

Lu, H., Zhang, X., & Yang, S. (2019). A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*.

Ma, Q., Ge, S., He, D., Thaker, D., & Drori, I. (2019). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. arXiv preprint arXiv:1911.04936.

Miki, S., Yamamoto, D., & Ebara, H. (2018). Applying deep learning and reinforcement learning to traveling salesman problem. In *2018 international conference on computing, electronics & communications engineering* (pp. 65–70). IEEE.

Nazari, M., Oroojlooy, A., Snyder, L., & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in neural information processing systems* (pp. 9839–9849).

Optimization, G. (2018). Gurobi optimizer reference manual. URL: http://www.gurobi.com.

Ouyang, W., Wang, Y., Han, S., Jin, Z., & Weng, P. (2021). Improving generalization of deep reinforcement learning-based tsp solvers. In *2021 IEEE symposium series on computational intelligence* (pp. 01–08). IEEE.

Pan, W., & Liu, S. Q. (2022). Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Applied Intelligence*, 1–18.

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Prates, M., Avelar, P. H., Lemos, H., Lamb, L. C., & Vardi, M. Y. (2019). Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 33* (pp. 4731–4738).

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, *211*(3), 427–441.

Satyananda, D., & Abdullah, A. (2021). Deep learning to handle congestion in vehicle routing problem: A review. In *Journal of physics: conference series, Vol. 2129*. IOP Publishing, Article 012023.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Sultana, N., Chan, J., Sarwar, T., & Qin, A. (2022). Learning to optimise general tsp instances. *International Journal of Machine Learning and Cybernetics*, 1–16.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT Press.

Vasek Chvatal, Applegate, D. L., Bixby, R. E., & Cook, W. J. (2006). Concorde tsp solver. URL: www.math.uwaterloo.ca/tsp/concorde.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. *Computer Science*, *28*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256.

Woeginger, G. J. (2003). Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization eureka, you shrink!* (pp. 185–207). Springer.

Xin, L., Song, W., Cao, Z., & Zhang, J. (2021). NeuroLKH: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, *34*, 7472–7483.

Xing, Z., Tu, S., & Xu, L. (2020). Solve traveling salesman problem by Monte Carlo tree search and deep neural network. arXiv preprint arXiv:2005.06879.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., et al. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of AAAI*.