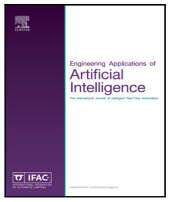
Engineering Applications of Artificial Intelligence 112 (2022) 104848

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



A graph convolutional encoder and multi-head attention decoder network for TSP via reinforcement learning

Jia Luo [a], Chaofeng Li [a,*], Qinqin Fan [a], Yuxin Liu [b]

[a] Institute of Logistics Science and Engineering, Shanghai Maritime University, 200135, Shanghai, China
[b] College of Information Engineering, Shanghai Maritime University, 200135, Shanghai, China

ARTICLE INFO

Keywords:
TSP
Graph convolutional network
Attention mechanism
Deep reinforcement learning

ABSTRACT

For the traveling salesman problem (TSP), it is usually hard to find a high-quality solution in polynomial time. In the last two years, graph neural networks emerge as a promising technique for TSP. However, most related learning-based methods do not make full use of the hierarchical features; thereby, resulting in relatively-low performance. Furthermore, the decoder in those methods only generates single permutation and needs additional search strategies to improve the permutation, which leads to more computing time. In this work, we propose a novel graph convolutional encoder and multi-head attention decoder network (GCE-MAD Net) to fix the two drawbacks. The graph convolutional encoder realizes to aggregate neighborhood information through updated edge features and extract hierarchical graph features from all graph convolutional layers. The multi-head attention decoder takes the first and last selected node embeddings and fused graph embeddings as input to generate probability distributions of selecting next unvisited node in order to consider global features. The GCE-MAD Net further allows to choose several nodes at each time step and generate a permutations pool after decoding to increase diversity of solution space. To assess the performance of GCE-MAD Net, we conduct experiments with randomly generated instances. The simulation results show the proposed GCE-MAD Net outperforms the traditional heuristics methods and existing learning-based algorithms on all evaluation metrics. Especially, when encountering large scale problem instances, the small scale pretrained GCE-MAD Net can get much better solutions than CPLEX solver with less time.

1. Introduction

Combinatorial Optimization (CO) problems have always gained widespread attention in applied mathematics and operations research, and exit in many real-life industries such as manufacturing, supply chain management, urban transportation, and lately in drone routing (Davendra, 2010; MirHassani and Habibi, 2013; Huang et al., 2020; Tran et al., 2020). Although wide research papers present new approaches to this field, it is still a challenge to obtain satisfactory results due to the NP-hardness of those CO problems especially in practical application scenarios (Paschos, 2014). The Traveling Salesman Problem (TSP) is among the most extensively solved CO problems in practice, and has been studied for its simple problem description (Hromkovič, 2013; Osaba et al., 2020). The state-of-the-art methodologies to TSP could be classified into exact methods, approximation methods, and heuristics methods that either require too much time to compute or not mathematically well defined. Exact methods can find the optimal solution under the theoretical guarantee, e.g., branch-and-bound (B&B) framework (Wang et al., 2012; Subramanyam and Gounaris, 2016; Kinable et al., 2017), but poorly on large-scale routing problems for their exponential complexity in the worst case. For some specific problems, approximation methods (Williamson and Shmoys, 2011) can find sub-optimal solutions with probable worst-case guarantees in polynomial time, but still be of poor approximation ratios (Rego et al., 2011). Although, heuristics can find satisfactory results within reasonable computational time, they lack theoretical guarantee on the solution quality, require substantial trial-and-error and highly depend on the intuition and experience of human experts to improve solution quality (Khan and Maiti, 2019; Pandiri and Singh, 2019; Ebadinezhad, 2020; Al-Gaphari et al., 2021; Saji and Barkatou, 2021).

To make a better trade-off between a good quality solution and a brief solving time to solve TSP, the learning-based methods, have been investigated and achieve competitive performance to the above non-learning-based methods (Bengio et al., 2021). The first challenge is to introduce learning-based algorithm for TSP. Vinyals et al. (2015b), for the first time, proposed a Ptr-Net for TSP with Recurrent Neural Networks (RNNs) which is trained by supervised learning and achieved significant improvement over no-learning-based methods on computing-time. However, it is hard to obtain the label dada when the

* Corresponding author.
E-mail address: wxlichaofeng@126.com (C. Li).

https://doi.org/10.1016/j.engappai.2022.104848
Received 18 September 2021; Received in revised form 20 January 2022; Accepted 22 March 2022
Available online 7 April 2022

number of nodes becomes large. To ease the difficulty of training the model without label data, Bello et al. (2016) further applied reinforcement learning to train the Ptr-Net for TSP. Although RL pretraining updated the model parameters with the actor–critic algorithm (Mnih et al., 2015), it failed to utilize the graph-structured features of TSP instances which can be embedded in node representations for different downstream tasks by graph embedding or network embedding techniques (Goyal and Ferrara, 2018). As a result, the pretrained model cannot make full use of node features.

What is more, same or similar actions would be taken according to the policy at decoding steps. Multiple decoders can generate different subsequences, which contributes to better complete solution. However, the existing learning-based methods, e.g. AM (Kool et al., 2018), S2V-DQN (Dai et al., 2017), Ptr-Net (Vinyals et al., 2015b), neglect to pursue more sequences when executing decoding strategy. Although Joshi et al. (2019) proposed a model which can output TSP solutions in one shot, but it needed additional procedures, such as beam search, to generate reasonable solutions when the period of training model is finished. Those additional procedures are used to find the optimal solution based on the same pretrained model, which will also reduce the diversity of the solution space.

To tackle the issues and limitations above, we propose a novel graph convolutional encoder and multi-head attention decoder network (GCE-MAD Net) by extracting the hierarchical features from the original TSP graph input and decoding multiple sequences to increase the diversity of solution space. The encoder based on GCN with node and edge features as input. The node features are 2-dimensional city coordinates and edge features are binary elements about any two cities connecting or not. The outputs of encoder are then passed into a decoder using attention mechanism (Vaswani et al., 2017) to predict the probability distribution of unselected nodes. The multiple decoders scheme is utilized to increase diversity of solution space at each timestep, specifically, each decoder generates a TSP solution and the optimal solution is selected from those solutions. The entire encoder–decoder network is trained by an improved reinforce learning (Williams, 1992) algorithm.

We propose a novel graph convolutional encoder and multi-head attention decoder network (GCE-MAD Net) for TSP. The contributions of this work are as follows:

(1) We propose a graph convolutional network as an encoder to aggregate neighbor features of each node. The node and edge features affect each other. The relative weight between two neighboring nodes is computed by the edge features, and edge features are updated through two connected node features. Furthermore, shallow features from the original graph input are obtained through residual block, which results in each node can aggregate features from all graph convolutional layers.

(2) We propose a multiple decoders strategy which can generate several complete sequences at once. The probability distribution of selecting next node is calculated through the multi-head attention mechanism-based decoder, which take the graph features, the first selected node features and the last selected node features as input. Furthermore, the multiple decoders scheme realizes to select several nodes at each time step to produce several complete sequences, which increases diversity of solution space.

(3) We propose a tailored deep reinforcement learning-based algorithm is designed to train the GCE-MAD Net. During training, the baseline is fixed until a stronger baseline appeared. New baseline is the minimal cost of several solutions generated by the multiple decoders scheme at each epoch. This baseline updated policy can ensure GCE-MAD Net is always improved over itself.

(4) Our experiments show the GCE-MAD Net is efficient and has stronger generalization ability than the state-of-the-art learning-based algorithms.

This paper is organized as follows. First, related work about tackling CO problems by machine learning based algorithms is discussed in Section 2. Section 3 gives the definition of TSP and the Markov Decision Process of the GCE-MAD Net. Section 4 introduces the proposed GCE-MAD Ne in detail. Section 5 introduces the training method of the proposed deep reinforcement learning method. We evaluate GCE-MAD Net on TSP instances and compare it against state-of-the-art machine learning methods, optimal solver CPLEX and traditional heuristics in Section 6. Finally, conclusions and prospects are listed in Section 7.

2. Related work

Existing methods for TSP mainly include exact algorithms, approximation methods, and heuristics methods. Those methods can be concluded as model-based (Wu et al., 2019; Ali et al., 2020; Al-Gaphari et al., 2021; Kanna et al., 2021; Saji and Barkatou, 2021; Wang and Han, 2021) methods which need to build MIP model first and only fit to a specific instance. The details of those model-based methods for TSP are summarized in Davendra (2010). Here, we focus on the emerging learning-based methods (Bengio et al., 2021; Li et al., 2021; Talbi, 2021) which have achieved dramatic advantages against conventional model-based methods in solution quality and computing-time.

Recent success of applying deep learning to solve CO problems can be traced back to the pointer network (Ptr-Net) (Vinyals et al., 2015b), it took three challenging CO problems as sequence-to-sequence problems, and overcame a drawback that output length depends on input by a pointer. Ptr-Net is a variant of attention mechanism and uses attention as a probability distribution called ‘pointer’ to select an element of the input sequence as the output. Ptr-Net is trained by supervise learning and the ground-truth output permutations are given by the Concorde solver. Because Ptr-Net is sensitive and expensive to the quality of label data, Bello et al. (2016) created an actor–critic reinforcement learning based algorithm, in which Ptr-Net is the actor network, three other network modules consist of the Critic network. Although, Ptr-Net architecture can learn a good solution for CO problems, it does not reflect graph structure of CO problems. The original network (Vinyals et al., 2015b) is designed for NLP, not for TSP, so there is a limitation which neglects the permutation invariance of the input cities. The work Nazari (2018) presented a permutation-invariance encoder to let the network learn the input order invariance. Kool et al. (2018) did not use positional encoding in the Transformer (Vaswani et al., 2017), and produced resulting node embeddings which were invariant to the input order.

Considering no unique representation of a TSP graph, Graph Neural Networks (GNNs) have potential to play the role as an encoder because of their permutation-invariance and sparsity-awareness (Wu et al., 2020; Zhou, 2020). Nazaria et al. encodes CO problems by a structure2vec graph embedding network and constructs solutions incrementally (Dai et al., 2017). Replacing structure2vec graph embedding model, Graph convolutional networks (GCNs) (Duvenaud et al., 2015; Defferrard et al., 2016; Gehring et al., 2017; Marcheggiani and Titov, 2017; Chen et al., 2018; Li et al., 2018) play an important role to encode node representations for estimating the likelihood of whether a node is part of optimal solution. Deudon et al. (2018) took the graph attention network as the encoder to aggregate neighbors features of each node, and utilized the same PN for selecting the node inserted into the subtour. The Sinkhorn Policy Gradient (SPG) algorithm were proposed to learn policies on permutation matrices. One sinkhorn layer followed the GRU Cho et al. (2014) was used to produce continuous relaxations of permutation matrices.

Different with the above deep reinforcement learning methods, some researches achieved the parameters of the GCN encoder to generate node embeddings for downstream task, i.e., large number of ground-truth output permutations were needed in advance to optimize the parameters. The graph learning network (GLN) directly learned the patterns of generated TSP solutions, in a certain sense, this model was trained by some ground-truth circles (Nammouchi et al., 2020). Joshi et al. (2019) took a GCN as the encoder to aggregate neighbors features,

and utilized a MLP to output the heatmap of possible connected edges of each node at one shot. Although two methods were trained by the supervised learning way, additional strategies, e.g. beam search, for searching optimal solution still increased the computing-time.

The above learning-based have achieved competitive performance on solving TSP, but most of them seldom reflects hierarchical features from the original graph input. Motivated by this, a residual graph convolutional network is used to exact and fuse features from all layers. Furthermore, single decoder in the existing learning-based methods is easy to make similar decision of choosing next node, so, we design a multiple decoders scheme to gain various probability distributions at each time step, which improves solution quality and enhance generalization ability.

3. Problem definition

In this paper, we focus on solving any random instance s which is the symmetric two-dimensional Euclidean TSP and formulated as an undirected graph $G(V, E)$. In the graph, $V = \{1, 2, \ldots, n\}$ (with $|V| = n$) represents a set of nodes, and $E = \{e_{11}, e_{12}, \ldots, e_{nn}\}$ denotes a set of edges, e_{ij} means relationship between node i and j. $\mathbf{x}_i \in \mathbb{R}^2$ is a vector representing coordinate of node i. Given coordinates of all nodes in the graph, $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, we wish to find the optimal permutation $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ with minimal tour length R. The elements $\pi_t \in V$ in permutation $\boldsymbol{\pi}$ selected at each time step $t \in \{1, \ldots, n\}$ are the orders of those nodes in the graph. Feasible permutation $\boldsymbol{\pi}$ must satisfy two conditions: (1) each node is served exactly once; (2) all nodes can only be served once, $\pi_t \neq \pi_{t'}, \forall t \neq t'$. The tour length of the feasible permutation $\boldsymbol{\pi}$ is defined as Eq. (1).

$$R(\boldsymbol{\pi}) = \left\|\mathbf{x}_{\pi_1} - \mathbf{x}_{\pi_n}\right\|_2 + \sum_{t=1}^{n} \left\|\mathbf{x}_{\pi_t} - \mathbf{x}_{\pi_{t-1}}\right\|_2 \tag{1}$$

Given a random TSP instance s, a stochastic policy $p_\theta(\boldsymbol{\pi}|s)$ used to generate a permutation $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ is defined as:

$$p_\theta(\boldsymbol{\pi}|s) = \prod_{t=1}^{n} p_\theta\left(\pi_t|s, \boldsymbol{\pi}_{1:t-1}\right) \tag{2}$$

here, θ represents parameters to be learned.

The iterative process to select node is modeled as the following Markov Decision Process (MDP).

(1) Observation $\boldsymbol{\pi}_{1:t}$ represents the generated subtour $\boldsymbol{\pi}_{1:t} = (\pi_1, \ldots, \pi_t)$ at time step $t \in \{1, \ldots, n\}$.

(2) Action a_t defines one node to be inserted in the subtour at time step $t \in \{1, \ldots, n\}$.

(3) Transition function $l\left(\boldsymbol{\pi}_{1:t}, a_t\right)$ converts observation $\boldsymbol{\pi}_{1:t}$ to $\boldsymbol{\pi}_{1:t+1}$, i.e., $\boldsymbol{\pi}_{1:t+1} = l\left(\boldsymbol{\pi}_{1:t}, a_t\right)$.

(4) Reward function is defined as Eq. (3).

$$r_t = r\left(\boldsymbol{\pi}_{1:t}, a_t, \boldsymbol{\pi}_{1:t+1}\right) = R\left(\boldsymbol{\pi}_{1:t}\right) \tag{3}$$

4. Proposed GCE-MAD Net

The details of proposed GCE-MAD Net are explained in the following subsections in terms of encoder architecture, decoder architecture. As visualized in Fig. 1, the architecture of the GCE-MAD Net follows the so-called encoder–decoder paradigm. In the figure, the GCN block consists of several graph convolutional layers, which get stacked on top of each other. After aggregating node features by the last graph convolutional layer, a mean pooling strategy is used to generate graph-level representation based on node embeddings. At each time step, in multi-head attention layer, the tailored query come from graph-level representation and node representations of the first selected node and the last selected node. The keys and values come from node representations. The query in single-head attention layer is the output of the multi-head attention layer, the keys also come from node representations. The output of the single-head attention layer is the compatibility of the query with all nodes, and it can be used to obtain the probability distribution of picking next unvisited node by softmax function. It is notable, the multiple decoder strategy in this paper allows to select several nodes for different permutations at a time, and the optimal solution comes from shoes permutations.

4.1. The encoder

Our encoder is based on GCN which exploits neural network operations over graph-structured data. For TSP instances, the input of the encoder includes two parts: node and edge features. Node feature $\mathbf{x}_i \in [0, 1]^2$ is a vector representing 2-dimensional coordinate of the ith node. Meanwhile, edge feature is a binary element about node i and j connecting or not and is defined by Eq. (4).

$$e_{ij} = \begin{cases} 1, \text{ if node } i \text{ connects with node } j \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

Because pairwise computation for all nodes is intractable when generalizing model to large-scale problem instances, k-nearest neighbors is adopted to make input graph sparse. Specifically, the neighbors of each node Ne are computed by Eq. (5) which realizes to diffuse information with the same message steps in different graph sizes.

$$Ne = n \times k\% \tag{5}$$

where, k is a hyperparameter that is a multiple of 10.

The input node and edge features are firstly respectively embedded in h dimensional features through two fully connected layers, and this operation is called 'primitive embedding' and represented by Eqs. (6) and (7).

$$\mathbf{h}_i^0 = \mathbf{A}_0\mathbf{x}_i + \mathbf{b}_0, \forall i \in \{1, \ldots, n\} \tag{6}$$

$$\mathbf{e}_{ij}^0 = \mathbf{A}_1\mathbf{e}_{ij} + \mathbf{b}_1, \forall j \in \{1, \ldots, n\} \tag{7}$$

where, $\mathbf{A}_0 \in \mathbb{R}^{h\times 2}$ and $\mathbf{A}_1 \in \mathbb{R}^h$ represent learnable weight parameters, $\mathbf{b}_0 \in \mathbb{R}^h$ and $\mathbf{b}_1 \in \mathbb{R}^h$ are defined as the bias parameters. The primitive node representation $\mathbf{h}_i^0$ and edge representation $\mathbf{e}_{ij}^0$ are passed into the first graph convolutional layer of the encoder. In the remaining section, $\mathbf{h}_i^l$ and $\mathbf{e}_{ij}^l$ denote node and edge representations of graph convolutional layer $l \in \{1, \ldots, L\}$ in the encoder, respectively, and both representations are alternatively updated as Fig. 2.

Fig. 2 describes a single graph convolutional layer to update node representations. The graph convolutional layer is regarded as a message passing process that information can be passed from one node to one of its neighbors with a certain probability. The probabilities are computed by the additional edge representations and they are summed up to one over all neighbors of node i. Residual connection is also applied to memorize information over each graph convolutional layer (Bresson and Laurent, 2017). As a result, the next graph convolutional layer node states derived by Eqs. (8) and (9). $\mathbf{h}_i^0$ and $\mathbf{e}_{ij}^0$ are the output of 'primitive embedding', which are calculated by Eqs. (6) and (7).

$$\begin{aligned} \mathbf{h}_i^l &= \mathbf{h}_i^{l-1} + \mathrm{ReLU}\left(\mathrm{BN}\left(\mathbf{W}_1^{l-1}\mathbf{h}_i^{l-1} + \sum_{j\in\mathbb{N}(i)} \boldsymbol{\eta}_{ij}^{l-1} \odot \mathbf{W}_2^{l-1}\mathbf{h}_j^{l-1}\right)\right), \\ \boldsymbol{\eta}_{ij}^{l-1} &= \frac{\sigma\left(\mathbf{e}_{ij}^{l-1}\right)}{\sigma\left(\sum_{j\in\mathbb{N}(i)} \mathbf{e}_{ij}^{l-1}\right)} \end{aligned} \tag{8}$$

$$\mathbf{e}_{ij}^l = \mathbf{e}_{ij}^{l-1} + \mathrm{ReLU}\left(\mathrm{BN}\left(\mathbf{W}_3^{l-1}\mathbf{h}_i^{l-1} + \mathbf{W}_4^{l-1}\mathbf{h}_j^{l-1} + \mathbf{W}_5^{l-1}\mathbf{e}_{ij}^{l-1}\right)\right), j \in \mathbb{N}(i) \tag{9}$$

where, $\mathbf{W}_1^{l-1}$, $\mathbf{W}_2^{l-1}$, $\mathbf{W}_3^{l-1}$, $\mathbf{W}_4^{l-1}$ and $\mathbf{W}_5^{l-1}$ are $\mathbb{R}^{h\times h}$ weight matrices to be learned, $\boldsymbol{\eta}_{ij}^{l-1}$ is a weight function to compute the relative weight between two neighboring nodes, $\sigma(\cdot)$ is the sigmoid function, $\mathrm{ReLU}(\cdot)$ is the rectified linear unit, $\mathrm{BN}(\cdot)$ stands for batch normalization.

Because stacking multiple graph convolutional layers can capture long-range dependencies among the graph-structured data, the encoder

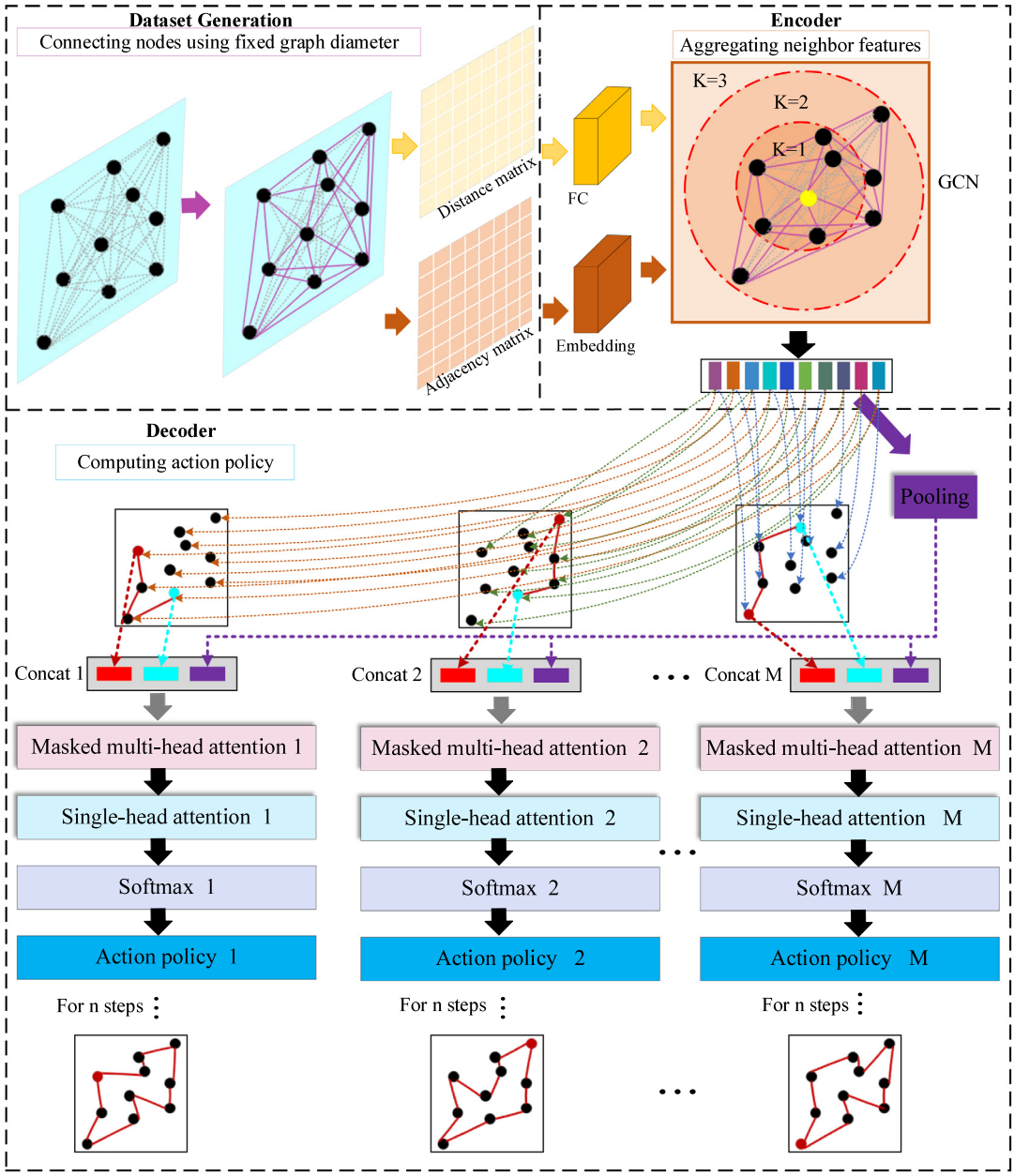


Fig. 1. The framework of proposed GCE-MAD Net.

adopts more than one graph convolutional layers and its structure is depicted as Fig. 3. Firstly, the primitive embedding module receives node features $\mathbf{x}_i \in \mathbb{R}^2$ and edge features $e_{ij} \in \mathbb{R}$ and projects them from low dimension to h dimension. Next, those h dimensional representations are passed to graph convolutional layers to update node representations. Specifically, the lth graph convolutional layer takes the output of $(l-1)$th graph convolutional layer as input. Finally, the output of the Lth graph convolutional layer is the output of the encoder, and is the updated node representations $\mathbf{h}_i^L$ with h dimension.

4.2. The decoder

The multiple decoders scheme represents more than one decoder with identical structures but unshared parameters, and the decoding procedure of single decoder follows Kool et al. (2018). Let $m \in \{1, \ldots, M\}$ be the index of decoder (the superscript (m) in following work means the index of decoder), every decoder constructs a solution $\boldsymbol{\pi}^m$ sequentially. To produce probabilities of visiting each valid node at timestep $t \in \{1, \ldots, T\}$ efficiently, a tailored *context* node is designed to gather messages with node representations $\mathbf{h}_i^L$. The *context* vector $\mathbf{c}_t^m$ can be view as the embedding of the tailored *context* node and defined formally as follows:

$$\mathbf{c}_t^m = \begin{cases} \text{concat}\left(\bar{\mathbf{h}}, \mathbf{v}_1^m, \mathbf{v}_2^m\right), t = 1 \\ \text{concat}\left(\bar{\mathbf{h}}, \mathbf{h}_{\boldsymbol{\pi}_{t-1}^m}^L, \mathbf{h}_{\boldsymbol{\pi}_1^m}^L\right), t > 1 \end{cases} \tag{10}$$

here $\text{concat}(\cdot)$ is the horizontal concatenation operator. The *context* vector $\mathbf{c}_t^m \in \mathbb{R}^{3 \cdot h}$ is calculated through concatenating the mean of node embeddings $\bar{\mathbf{h}} = \frac{1}{n}\sum_{i=1}^n \mathbf{h}_i^L$, embedding of the last selected node $\mathbf{h}_{\boldsymbol{\pi}_{t-1}^m}^L$ and embedding the first selected node $\mathbf{h}_{\boldsymbol{\pi}_1^m}^L$. For $t = 1$, we use learned parameters $\mathbf{v}_1^m \in \mathbb{R}^h$ and $\mathbf{v}_2^m \in \mathbb{R}^h$ as input placeholders of each decoder.

Following multi-head attention mechanism (Vaswani et al., 2017), a query and a set of key–value pairs are needed to map the output.

$$\mathbf{q}_t^m, \mathbf{k}_i^m, \mathbf{v}_i^m = \mathbf{W}_Q^m \mathbf{c}_t^m, \mathbf{W}_K^m \mathbf{h}_i^L, \mathbf{W}_V^m \mathbf{h}_i^L \tag{11}$$

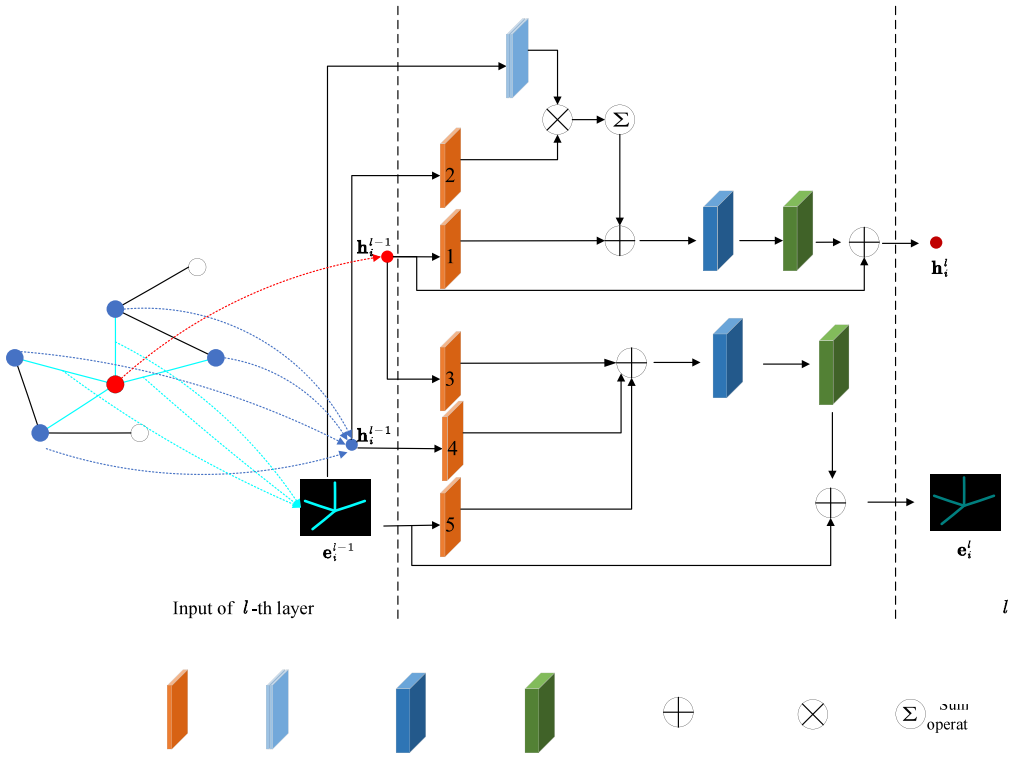


Fig. 2. Updating procedure of node representations (Graph Convolutional Layer).

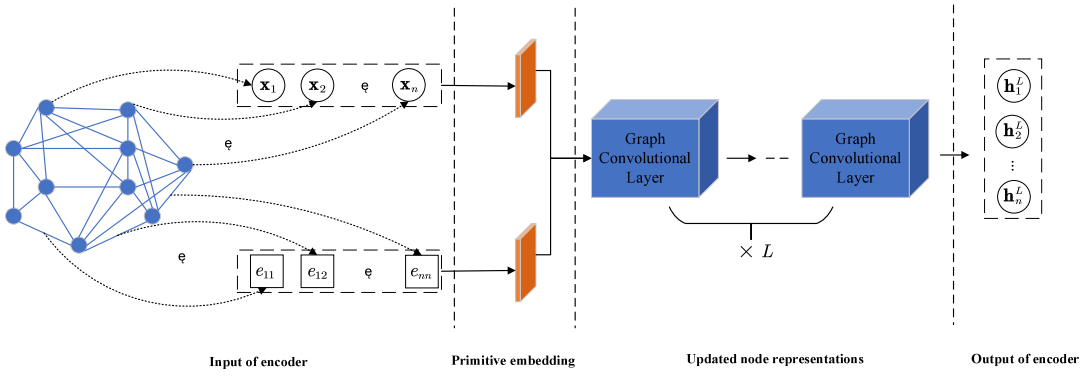


Fig. 3. The structure of the encoder.

here, parameters $\mathbf{W}_Q^m$ is a $\mathbb{R}^{k\times 3h}$ matrix and $\mathbf{W}_K^m$ is a $\mathbb{R}^{k\times h}$ matrix, $k=\frac{h}{F}$, F is the number of heads, $\mathbf{W}_V^m$ is a $\mathbb{R}^{v\times h}$ matrix. In this work, keys $\mathbf{k}_i^m \in \mathbb{R}^k$ and values $\mathbf{v}_i^m \in \mathbb{R}^v$ keep unchanged during decoding period, so, keys $\mathbf{k}_i^m$ and values $\mathbf{v}_i^m$ of each decoder are only computed as Eq. (11) once, but iterate a single query $\mathbf{q}_t^m$ from the *context* vector at every timestep t.

The compatibility $u_{j,t}^m \in \mathbb{R}$ of query $\mathbf{q}_t^m$ with all nodes are computed according to Eq. (12). For TSP, masking nodes when it is visited, i.e., setting $u_{j,t}^m = -\infty$ at time step t:

$$u_{j,t}^m = \begin{cases} \frac{(\mathbf{q}_t^m)^T \mathbf{k}_j^m}{\sqrt{k}}, \text{if } j \neq \pi_{t'}^m, \forall t \neq t' \\ -\infty, \text{otherwise} \end{cases} \tag{12}$$

Next, we can get the vector $\mathbf{a}_t^m$ through convex combination of messages $\mathbf{v}_j^m$ at timestep t:

$$\mathbf{a}_t^m = \sum_j \text{softmax}\left(u_{j,t}^m\right) \mathbf{v}_j^m \tag{13}$$

As we take the multi-head attention mechanism, let we denote the above result vector by $\mathbf{a}_t^{m,f}$ for $f \in \{1,2,\ldots,F\}$. Using learnable matrix $\mathbf{W}_O^{m,f} \in \mathbb{R}^{h\times v}$ can project back to a vector $\mathbf{c}_t^{m*} \in \mathbb{R}^h$. The final multi-head attention value for *context* node is calculated as Eq. (14).

$$\mathbf{c}_t^{m*} = \sum_{f=1}^{F} \mathbf{W}_O^{m,f} \mathbf{a}_t^{m,f} \tag{14}$$

Finally, to compute probability distribution, a layer with single attention head is added on the layer with multi-heads. Following Bello et al. (2016), we clip the compatibilities within $[-D, D]$ using tanh:

$$u_{j,t}^m = \begin{cases} D^* \tanh\left(\frac{(\mathbf{r}_t^m)^T \mathbf{g}_j^m}{\sqrt{k}}\right), \text{if } j \neq \pi_{t'}^m, \forall t \neq t' \\ -\infty, \text{otherwise} \end{cases} \tag{15}$$

$$\mathbf{r}_t^m, \mathbf{g}_t^m = \mathbf{W}_R^m \mathbf{c}_t^{m*}, \mathbf{W}_G^m \mathbf{h}_j^L \tag{16}$$

here, parameters $\mathbf{W}_R^m$ and $\mathbf{W}_G^m$ are $\mathbb{R}^{h\times h}$ matrices due to a single head, i.e., $k=\frac{h}{F}$ and $F=1$.

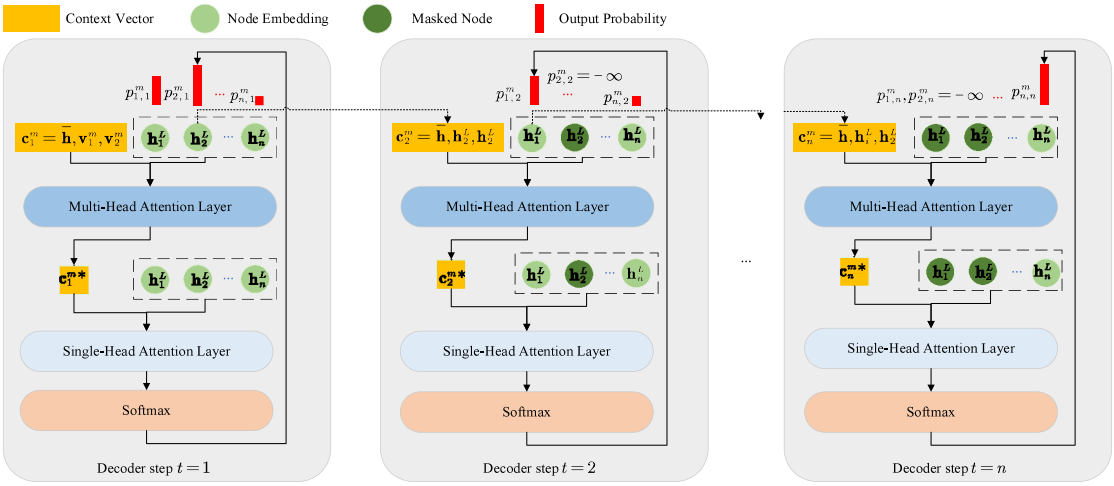


Fig. 4. The decoding procedure for TSP.

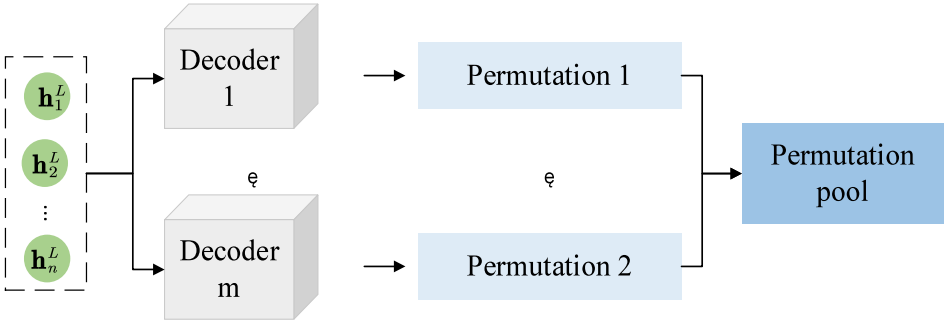


Fig. 5. The structure of the decoder.

The compatibilities computed through Eq. (15) as unnormalized log-probabilities is used to compute the probability vector $\mathbf{p}^m$ by softmax function. The elements in $\mathbf{p}^m$ are defined by Eq. (17).

$$p^m_{i,\mathrm{t}} = p_\theta\left(\pi^m_t = i|s, \boldsymbol{\pi}^m_{1:t-1}\right) = \mathrm{softmax}\left(u^m_{j,t}\right) \tag{17}$$

The process of single decoder to pick nodes is described as Fig. 4. At each time step t, the multi-head attention layer takes the *context* vector $\mathbf{c}^m_t$ and node embeddings $\mathbf{h}^L_i$ as input, and outputs the final multi-head attention value $\mathbf{c}^{m*}_t$ for the *context* node. Then, the single-head attention layer takes the vector $\mathbf{c}^{m*}_t$ and node embeddings $\mathbf{h}^L_i$ as input to generate the compatibilities. The probability distribution of selecting the next valid node is derived by Eq. (17).

Fig. 5 shows the structure of the decoder, 'Decoder' module in it has been described in Fig. 4. Each decoder can generate a permutation, the final solution is selected from those permutations.

5. Training method

For random input instance s, each decoder individually samples a trajectory $\boldsymbol{\pi}^m$ to get separate REINFORCE loss with the same greedy rollout baseline. For training the model, we define the sum of the expectation of the cost $R(\boldsymbol{\pi}^m)$ (tour length computed as Eq. (1)) as the loss function presented by Eq. (18).

$$loss(\boldsymbol{\theta}|s) = \sum_m E_{p_\theta(\boldsymbol{\pi}|s)}\left[R(\boldsymbol{\pi}^m)\right] \tag{18}$$

where, parameters $\boldsymbol{\theta}$ is optimized by gradient descent using the REINFORCE algorithm with rollout baseline $b(s)$.

$$\nabla_\theta loss(\boldsymbol{\theta}|s) = \sum_m E_{p_\theta(\boldsymbol{\pi}|s)}\left[\left(R(\boldsymbol{\pi}^m) - b(s)\right)\nabla_\theta \log p_\theta(\boldsymbol{\pi}^m|s)\right] \tag{19}$$

The model with the best set of parameters among previous epochs is used as the baseline model to greedily decode the result as the baseline $b(s)$. The value of baseline $b(s)$ in this work is the minimal cost defined as Eq. (20).

$$b(s) = \min_m R\left(\boldsymbol{\pi}^m = \pi^m_1, \ldots, \pi^m_n|s\right) \tag{20}$$

$$\pi^m_t = \underset{i}{\mathrm{argmax}}\left(p_\theta\left(\pi^m_t = i|s, \boldsymbol{\pi}^m_{1:t-1}\right)\right) \tag{21}$$

All steps are listed as Algorithm 1.

6. Experiments

Controlled experiments are designed to probe the performance of the GCE-MAD Net on solving 2D Euclidean TSP instances.

6.1. Evaluation metrics

Five typical metrics are used to evaluate the performance of GCE-MAD Net, they are defined as follows.

(1) Average predicted tour length. The average predicted tour length over validation and test instances, computed as Eq. (22)

$$\overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \tag{22}$$

(2) Win rate. The win rate r_{win} is the proportion of winning cases over the total number of tested cases N, as defined at Eq. (23). A case is considered won when its tour length computed by a method is shorter than the tour length computed by the CPLEX.

$$r_{win} = N_{win}/N \tag{23}$$

where N_{win} is the number of tested winning cases.

Algorithm 1 REINFORCE with Rollout Baseline

Input: number of decoders M ; number of epochs E ; steps per epoch T ; instance size B ; significance level α ; actor networks with trainable parameters $\boldsymbol{\theta}$; Baseline policy network with trainable parameters $\boldsymbol{\theta}^b$.

Init $\boldsymbol{\theta}$, $\boldsymbol{\theta}^b$
for epoch = $1,2,...,E$ **do**
  for step = $1,2,...,T$ **do**
    $s_i \leftarrow$ Randomly generate instances, $\forall i \in \{1,2,...,B\}$
    $\left\{\boldsymbol{\pi}_i^1,...,\boldsymbol{\pi}_i^M\right\} \leftarrow p_\theta\left(\pi_t^m \middle| s_i, \boldsymbol{\pi}_{1:t-1}^m\right), \forall i \in \{1,...,M\}, m \in \{1,...,M\}$
    $\left\{\boldsymbol{\pi}_i^{b,1},...,\boldsymbol{\pi}_i^{b,M}\right\} \leftarrow p_{\theta^b}\left(\pi_t^m \middle| s_i, \boldsymbol{\pi}_{1:t-1}^m\right), \forall i \in \{1,...,M\}, m \in \{1,...,M\}$
    $b(s) = \min\left(R\left(\boldsymbol{\pi}_i^{b,m}\right)\right), \forall i \in \{1,...,M\}, m \in \{1,...,M\}$
    $\nabla loss \leftarrow \sum_{i=1}^{B}\sum_{m=1}^{M}\left(R\left(\boldsymbol{\pi}_i^m\right) - b(s)\right)\nabla \log p_\theta\left(\boldsymbol{\pi}_i^m\right)$
    $\boldsymbol{\theta} \leftarrow \mathrm{Adam}\left(\boldsymbol{\theta}, \nabla loss\right)$
  end for
  if Paired t-test $\left(p_\theta, p_{\theta^b}\right) < \alpha$ **then**
    $\boldsymbol{\theta}^b \leftarrow \boldsymbol{\theta}$
  end if
end for
Return trained parameter set $\boldsymbol{\theta}$

(3) Fail rate. The fail rate r_{fail} is the proportion of failed cases over the total number of tested cases N, as defined at Eq. (24). A case is considered failed when its tour length computed by a method is longer than the tour length computed by the CPLEX.

$$r_{fail} = N_{fail}/N \tag{24}$$

where N_{fail} is the number of tested winning cases.

(4) Optimality gap. The average optimal gap between optimization solver CPLEX and our model is computed as Eq. (25).

$$Gap = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{R_i'}{R_i} - 1\right) \tag{25}$$

here, N denotes the number of instances; R_i' is the optimal solution delivered by our model; R_i means the IBM CPLEX's optimal result.

(5) Evaluation time. The inference time of our model shown as Tables 2–4 is run on one single Nvidia Geforce 3090 GPU. The time of CPLEX and other heuristics solutions are gained on Intel(R) Core (TM) i9-10900 CPU @ 2.80 GHz.

6.2. Dataset generation

Training data. As described in our problem statement at Section 3, the 2D coordinates of each node is the only information should be known in advance. Following the way of generating dataset in Kool et al. (2018), for each node, we generate its coordinates by sampling uniformly at random in the unit square $[0,1] \times [0,1]$, i.e., problem instances consist of a set of 2D coordinates. It is notable that the value of coordinate retains 16 decimal places, which leads to $10^{16} \times 10^{16}$ different nodes in the unit square. The parameters of the GCE-MAD Net are trained for 100 epochs on training dataset which is generated **on the fly**. In each training epoch, 2500 batches of 512 instances are processed.

Validation and test data. Validation and test instances are generated separately in advance on the same data generator.

6.3. Training setting

The proposed approach is implemented using Pytorch with CUDA acceleration, specifically, we use a Nvidia Geforce 3090 GPU with 24 GB of memory to complete the training, and construct the model by Torch 1.7.0, and implement the code by Python 3.6.

Following Kool et al. (2018) and Joshi et al. (2019), our model is trained with the end-to-end manner using Adam, with constant learning rate of 1×10^{-4}, batch size of 512, and 100 epochs. We initialize the weights of our model using the uniform initializer $U\left(-1/\sqrt{d}, 1/\sqrt{d}\right)$, where d is the input dimension. The number of graph convolutional layers is 3, node and edge embedding dimension are both set 128, the graph diameter of each node equals 50%, which lead to a good trade-off between solution quality and computational complexity. The number of heads in the multi-head attention decoder is 8, and the number of decoders equals 5. Finally, significance in the t-test is 0.05.

For the limited GPU memory, the datasets generated on the fly of two problem sizes (20, 50) are used to train the parameters of the GCE-MAD Net, which generates two parameter sets of the GCE-MAD Net. For evaluating their performance, the test datasets of three problem sizes (20, 50 and 100) are generated. For readability, the details of those datasets in this work are listed in Table 1. The pretrained models on TSP20_Training and TSP50_Training sets are named Model_TSP20 and Model_TSP50 respectively.

Table 1
The details of data set.

Data	Name	Number
Training set	TSP20_Training	1,280,000
	TSP50_Training	1,280,000
Validation set	TSP20_Validation	1280
	TSP50_Validation	1280
Testing set	TSP20_Testing	1280
	TSP50_Testing	1280
	TSP100_Testing	1280

6.4. Sensitivity analysis

In this section, we analyze sensitivity of four parameters that could greatly influence the proposed model, i.e., the number of decoders, the number of attention heads in each decoder, the embedding dimension in GCN block, and the number of layers in the encoder. The line plots in Fig. 6 shows changing conditions of average tour length when we

Table 2
Comparison on instances with 20 customers.

Method		$\bar{R}$	r_{win}	r_{fail}	Gap	t (s)
MIP	CPLEX	3.89 + −0.32	–	–	0	18.12
Heuristic	Nearest neighbor	4.52 + −0.03	2.81%	97.19%	16.13%	2.78
	Nearest insertion	4.34 + −0.02	2.03%	97.97%	12.03%	0
	Random insertion	4.02 + −0.02	24.77%	75.23%	3.32%	0
	Farthest insertion	3.94 + −0.02	39.22%	60.78%	1.30%	0
Learning-based algorithm	AM	3.86 + −0.31	59.69%	26.40%	−0.71%	0.07
	GCN	3.93 + −0.33	31.32%	61.88%	1.24%	0.09
	GLN-TSP	3.85	–	–	–	–
	Our GCE-MAD Net	**3.85 + −0.31**	**71.41%**	12.73%	−0.91%	0.23

Table 3
Comparison on instances with 50 customers.

Method		$\bar{R}$	r_{win}	r_{fail}	Gap	t (s)
MIP	CPLEX	5.76 + −0.27	–	–	0.00%	2.78
Heuristic	Nearest neighbor	7.00 + −0.03	0.08%	99.92%	21.50%	0
	Nearest insertion	6.78 + −0.02	0	100%	17.84%	0
	Random insertion	6.13 + −0.02	2.9%	97.1%	6.48%	0
	Farthest insertion	6.01 + −0.02	9.92%	90.08%	4.36%	0.17
Learning-based algorithm	AM	5.78 + −0.28	39.38%	59.3%	0.35%	0.25
	GCN	6.08 + −0.32	19.92%	80.08%	4.86%	–
	GLN-TSP	5.85	–	–	–	0.70
	Our GCE-MAD Net	**5.73 + −0.27**	**58.75%**	40.31%	−0.40%	2.78

Table 4
Comparison on instances with 100 customers.

Method		$\bar{R}$	r_{win}	r_{fail}	Gap	t (s)
MIP	CPLEX	9.01 + −0.73	–	–	0.00%	121.17
Heuristic	Nearest neighbor	9.67 + −0.03	21.48%	78.52%	7.94%	2.68
	Nearest insertion	9.45 + −0.02	23.59%	76.41%	5.54%	0.01
	Random insertion	8.52 + −0.02	73.52%	26.48%	−4.93%	0
	Farthest insertion	8.35 + −0.02	81.80%	18.20%	−6.77%	0
Learning-based algorithm	AM	8.14 + −0.28	93.60%	6.40%	−9.19%	0.26
	GCN	8.78 + −0.34	57.81%	42.19%	−1.98%	0.60
	GLN-TSP	–	–	–	–	–
	Our GCE-MAD Net	**8.04 + −0.27**	**96.33%**	3.77%	−10.27%	0.28

change those parameters in our model. As we aim to find the optimal solution with minimal cost (tour length), the line plots should always present downward trend.

From Fig. 6(a), we find the dimension of embedding in the encoder can greatly influence our mode. Useful information of customer features in low dimensional space may be neglected, which can explain the improvement with increased embedding dimension. However, a little change happened when we set dimension from 128 to 256, because node and edge features of TSP are easy to be represent.

Similarly, we assess the performance of the number of embedding layers in the encoder, as Fig. 6(b). The improvement of increased layers explains GCNs aggregate and pass message by stacking layers.

Next, we retrain our model by setting the number of decoders in 1, 2, 3, 4 and 5. The performance curves are depicted in Fig. 6(c). It is obviously that better solutions can be found with increased decoders, especially, when adding the headers from 1 to 2.

Finally, the evaluation curves of multiple attention heads are plotted in Fig. 6(d). More attention heads allow *context* node to receive more types of messages from all nodes. Through those messages, we can select the next node inserting to the optimal solution, so more heads improve the optimal solution quality.

6.5. Baselines

The approaches selected for the performance comparison include two state-of-the-art deep learning methods (Kool et al., 2018; Joshi et al., 2019), two heuristics used to compare in Kool et al. (2018) and commonly-used solver CPLEX. The details on these approaches are introduced as following.

(1) Nearest neighbor

The nearest neighbor heuristic initializes a path with a random single node (we always start with the first node in the input). In each iteration, the next node is selected as it is the nearest one to the end node of the current partial path. The selected in this iteration becomes the new end node. Finally, after all nodes are selected, the end node should connect to the start node to form a tour.

(2) Nearest/random/farthest insertion

The insertion heuristics initialize a tour with two nodes. In each iteration, one node is selected to insert to the tour by some rules. Different rules generate different insertion heuristics. Let S be the set of nodes in the tour, d_{ij} represents the distance from node i to j. Nearest insertion inserts node i so that it can be nearest to any node in the tour:

$$i^* = \underset{i \notin S}{\operatorname{argmin}} \min_{j \in S} d_{ij} \tag{26}$$

Farthest insertion inserts node i so that the distance from node i to the nearest node j is maximal:

$$i^* = \underset{i \notin S}{\operatorname{argmax}} \min_{j \in S} d_{ij} \tag{27}$$

Random insertion inserts a node randomly.

6.6. Comparison with state-of-the-art methods

Tables 2–4 show the predicted tour length of each method under different instance sizes and each method is run on the same test dataset.

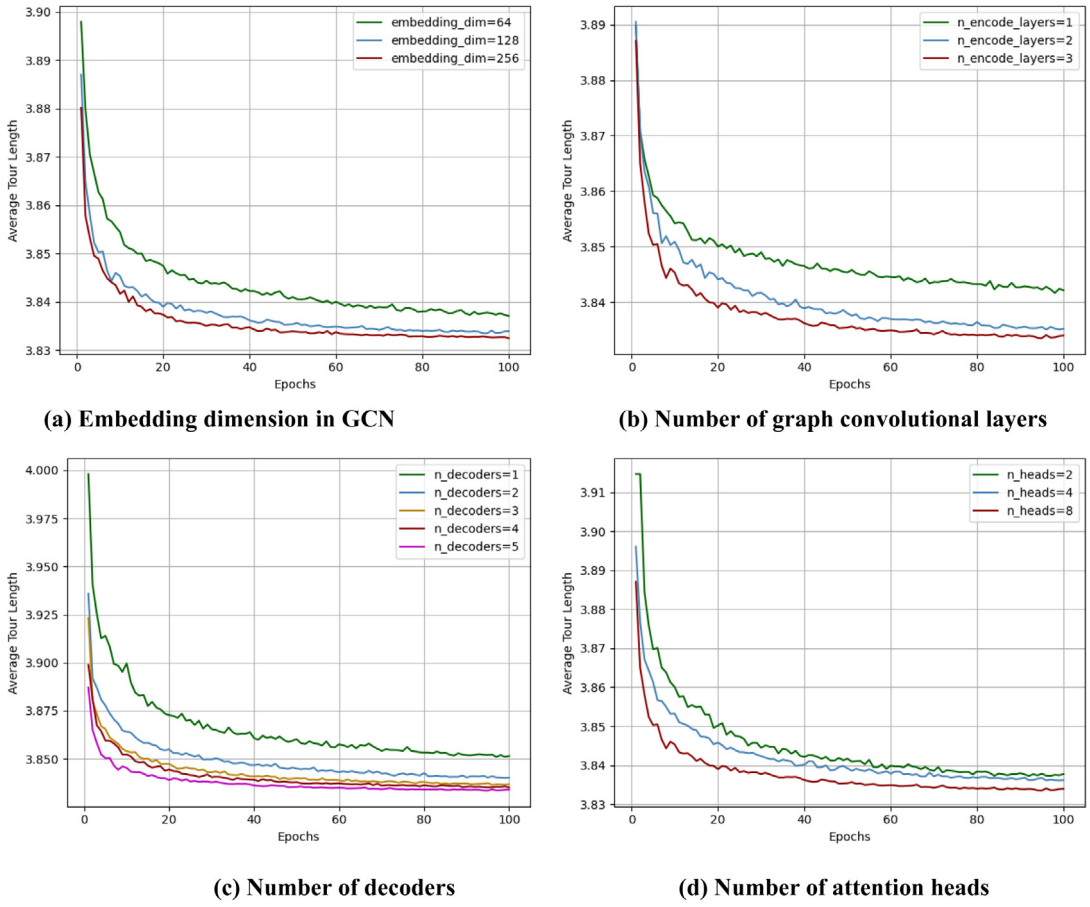


Fig. 6. Sensitivity analysis in GCE-MAD Net parameters.

For fair comparison, we take pretrained models from Kool et al. (2018) and GCN (Joshi et al., 2019), and both models have the same 100 trail runs. The results of GLN-TSP are the same with the original paper (Nammouchi et al., 2020). As shown in Tables 2–4, the results on 20 and 50 customers are obtained on Model_TSP20 and Model_TSP50 respectively. Note that our result on 100 customers is also obtained on Model_TSP50. As we can see, our GCE-MAD Net can get the minimal average tour length in test data among those classical heuristics and learning-based algorithms. It also outperforms the CPLEX, especially the performance on instances with 100 customers.

For clearly comparing the performance of the heuristics methods and the CPLEX on 1,280 test instances, except the indicator optimality gap, we also give the value of the win rate r_{win} and fail rate r_{fail}. When solving TSP20_Testing and TSP50_Testing, our GCE-MAD Net can find better solutions on over 50% instances than the CPLEX. It is notable that the GCE-MAD Net outperform the CPLEX on near 100% TSP100_Testing instances, which demonstrates that our GCE-MAD Net keeps a good competitive performance with increasing problem complexity.

Fig. 7 shows the situation that the farthest insertion, the AM (Kool et al., 2018), and the GCE-MAD Net compare with the CPLEX on solving TSP20_Testing, TSP50_Testing, and TSP100_Testing. The coordinates of a blue point represent the tour length computed by the three comparison methods and the tour length computed by the CPLEX. If the blue point is under the red line ($y = x$), it represents that the CPLEX finds a better solution than the comparison method, and other situation can be understood in the same way. Fig. 7 (a), (b), and (c) are the comparison of three methods and the CPLEX when they are used to solve TSP20_Testing, as we can see, our GCE-MAD Net find better solution than the CPLEX on most cases, only a few blue points are under the red line. Fig. 7 (d), (e), and (f) are the comparison of three methods and the CPLEX when they are used to solve TSP50_Testing, the farthest insertion behaves badly, because as Fig. 7 (d) shows that almost all blue points are under the red line. In contrast, two learning-based methods perform well, but the GCE-MAD Net outperforms the AM (Kool et al., 2018). Fig. 7 (g), (h), and (i) are the comparison of three methods and the CPLEX when they are used to solve TSP100_Testing. The CPLEX performs worse than three comparison methods on most cases, which also demonstrates that the traditional solver CPLEX is powerless on large scale problems, and our GCE-MAD Net is a nice alternative.

6.7. Impact of graph density

In this paper, we use a fixed graph diameter strategy defined as Eq. (5) to realize different graph sizes with different graph density, i.e., the neighbors of each node in different graph sizes should be various. Fig. 8 shows the change with different graph density. The pretrained Model_TSP20 and Model_TSP50 with different graph density are tested on three hold-out datasets: TSP20_Testing, TSP50_Testing, and TSP100_Testing. From Fig. 8, as we can see, no matter the pretrained Model_TSP20 or Model_TSP50, their performances are better with the graph of a sparse 40%-nearest neighbors than the full graph, i.e., no neighbors or too much neighbors will decrease the performance of the pretrained model.

6.8. Ablation study

In order to show the efficiency of the GCE-MAD Net, two integral parts of this network need to be test: (1) the residual block in the encoder, (2) the multi-decoder strategy. Three models with the corresponding part absent are trained, and their performance on the

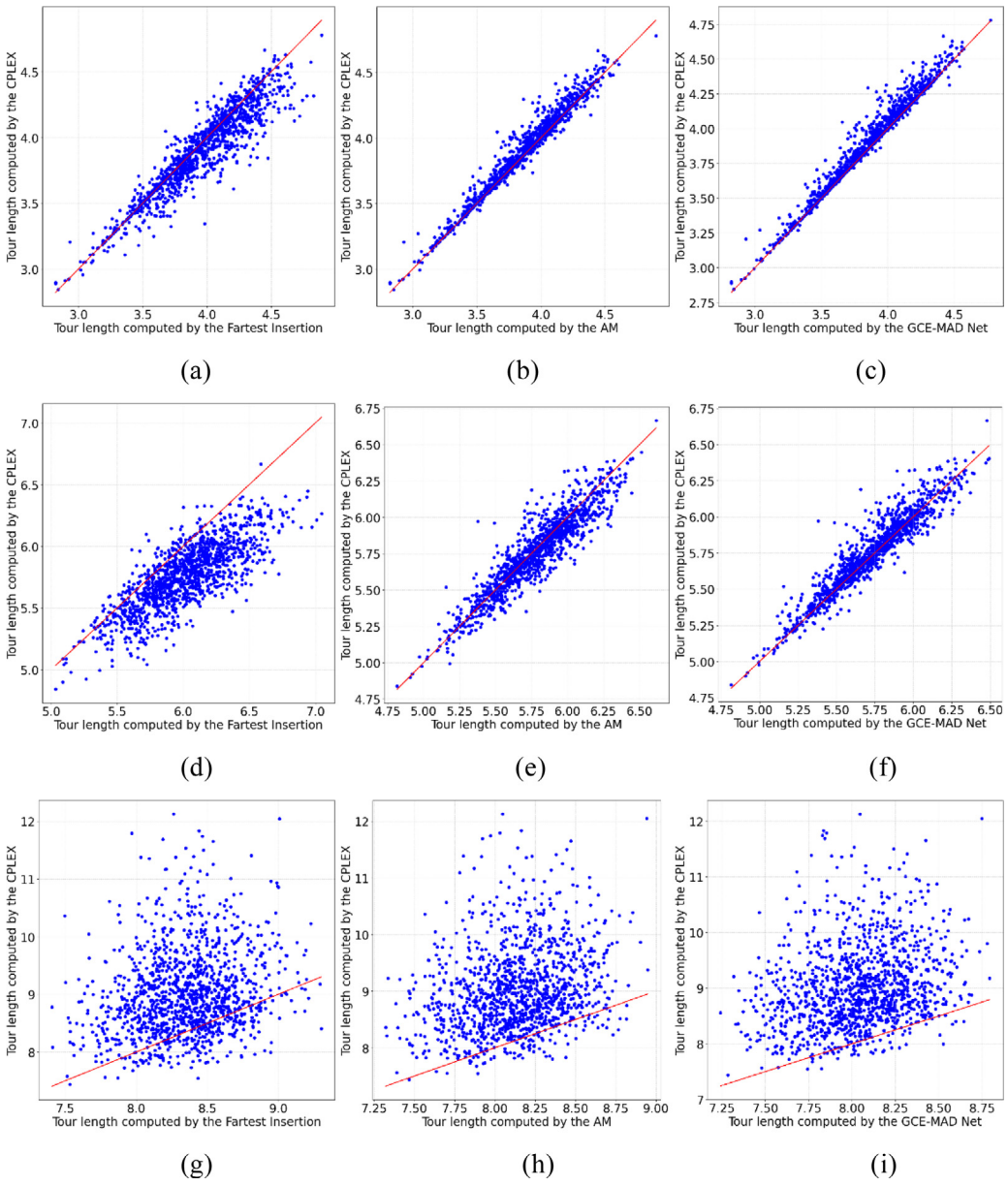


Fig. 7. The comparison of three methods (the farthest insertion, the AM (Kool et al., 2018), and our GCE-MAD Net) with the CPLEX on TSP20_Testing (Fig. 7 (a), (b), and (c)), TSP50_Testing (Fig. 7 (d), (e), and (f)), and TSP100_Testing (Fig. 7 (g), (h), and (i)).

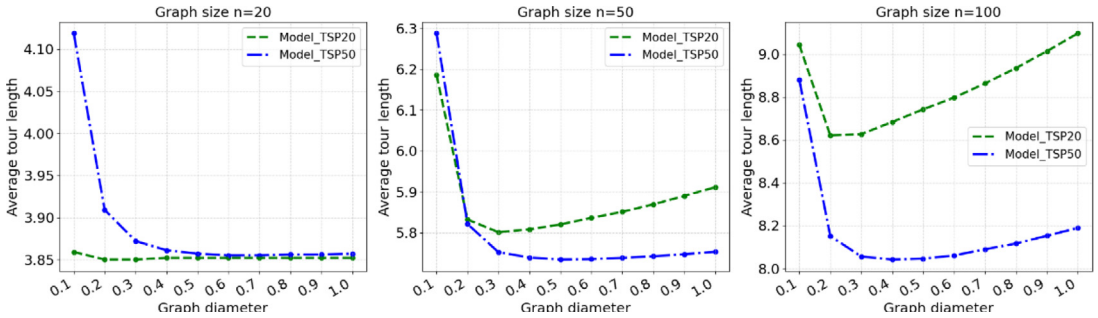
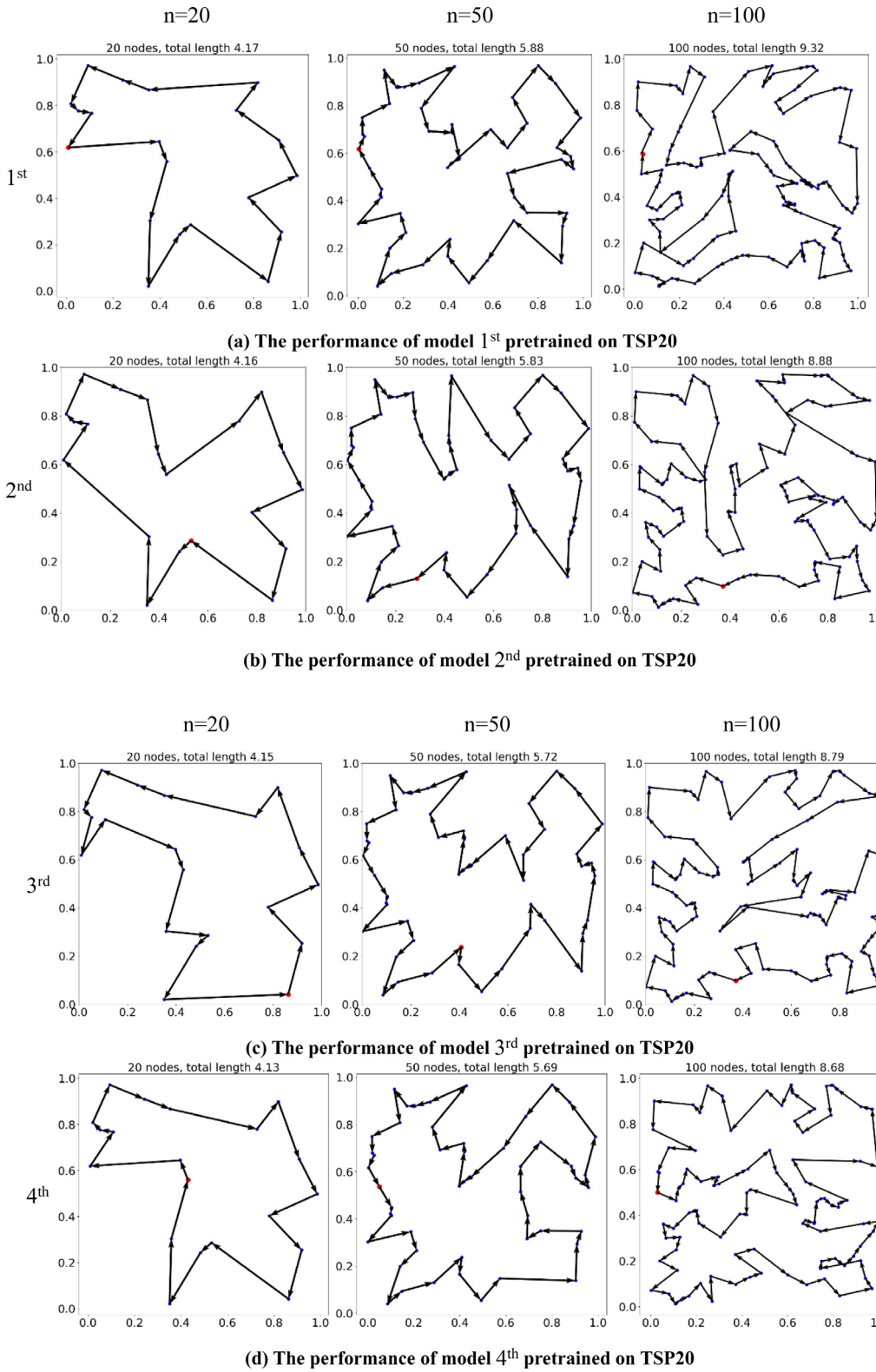


Fig. 8. The impact of different graph diameter.



(a) The performance of model 1st pretrained on TSP20

(b) The performance of model 2nd pretrained on TSP20

(c) The performance of model 3rd pretrained on TSP20

(d) The performance of model 4th pretrained on TSP20

Fig. 9. Visualization of random samples for ablation study.

Table 5
Ablation investigation of residual block and multi-decoder.

Model		1st	2nd	3rd	4th
Network settings	Residual block	✗	✓	✗	✓
	multi-decoder	✗	✗	✓	✓
Average tour length	20	3.867 + −0.315	3.865 + −0.314	3.854 + −0.310	3.850 + −0.309
	50	5.941 + −0.316	5.929 + −0.313	5.852 + −0.297	5.832 + −0.292
	100	9.055 + −0.392	9.028 + −0.415	8.641 + −0.333	8.620 + −0.336

same test dataset are used to compare to the complete model (the GCE-MAD Net). Details of those four models are listed as Table 5. It is notable that the four networks have the same graph diameter $k = 40\%$, and are pretrained on TSP20, then tested on TSP20, TSP50 and TSP100. The baselines of those problem sizes are obtained by model 1st without residual block and multi-decoder, which are very poor (the average tour length is 3.867 + −0.315, 5.941 + −0.316, 9.055 + −0.392, respectively). Then, one of residual block or multi-decoder is added to the baseline, resulting in 3.865 + −0.314, 5.929 + −0.313, 9.028 + −0.415 and 3.854 + −0.310, 5.852 + −0.297, 8.641 + −0.333 respectively (from model 2nd to 3rd in Table 5), which validates that each component can efficiently improve the performance of the baseline. Finally, we further add two components to the baseline, resulting in 3.850 + −0.309, 5.832 + −0.292, 8.620 + −0.336 (model 4th in Table 5). It can be concluded that two components perform better than only one component. We also visualize random samples of three problem sizes as Fig. 9. The 4th model (the GCE-MAD Net) performs the best, which demonstrates that the effectiveness and benefits of the residual block and multi-decoder strategy.

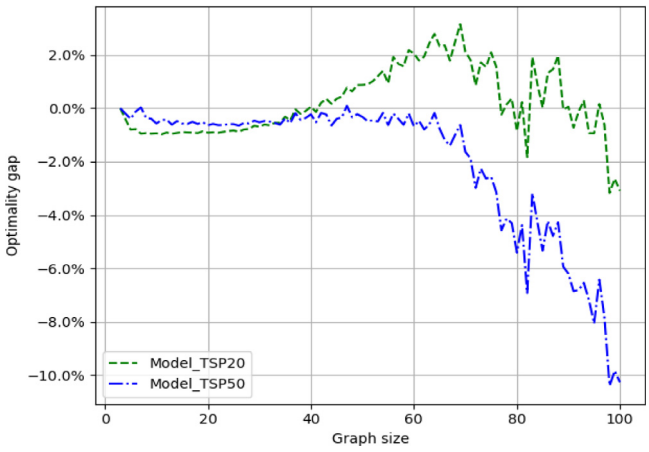


Fig. 10. The generalization performance.

6.9. Generalization ability of GCE-MAD Net

In the real world, delivery requests may come from over 50 customers, but training on larger graphs from scratch is intractable and sample inefficient, so, small-scale pretrained model generates well on large-scale problem instance is very important. We demonstrate the generalization ability of the proposed model in Fig. 10. A hold-out test out of 124,160 TSP instances, consist of 1280 instances each of TSP3, TSP4, ..., TSP100. Model_TSP20 and Model_TSP50 are two pretrained models which are trained on graph size 20 and 50 nodes respectively and are used to evaluate on variable sizes (from 3 to 100) instances. Optimality gaps of those instances are shown as Fig. 10. As it is difficult for CPLEX to find the best solution of large-scale instances, the gap shocks back and forth during graph size from 70 to 100. When graph size is from 4 to 40, Model_TSP20 and Model_TSP50 both outperform the CPLEX. Starting from graph size 40 nodes, Model_TSP50 still outperforms the CPLEX. In general, we can conclude that the GCE-MAD Net pretrained on larger graph size achieves better generalization ability.

7. Conclusion and future direction

In this research, a novel deep reinforcement learning based encoder–decoder framework called GCE-MAD Net is proposed to solve TSP. We propose a graph convolutional network to aggregate neighbors features of each node through edge features and fuse features from all graph convolutional layers. The attention decoder allows to select next city according to graph-level features, which is beneficial to take an action from global view. Furthermore, the multiple decoders strategy in this work results in several solutions at one time, which diversity the solution space. The computational experiments verify proposed GCE-MAD Net performs better than the state-of-the-art deep reinforcement learning based algorithms, also supreme over CPLEX optimizer and traditional heuristics. Our findings suggest that the deep reinforcement learning based algorithms are successful to solve CO problems. Unlike some traditional heuristics, our model performs well in solution quality and efficiency. Moreover, well-trained models can be used offline to find optimal solutions in very little time even with variable sizes customers.

In future research, extending the model to solve very large-scale TSP with a huge number of customers is of great interest. A more challenge task is to propose deep reinforcement learning based algorithms to solve more realistic problems, e.g., TSP with time window, dynamic demands and so on.

CRediT authorship contribution statement

Jia Luo: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Writing – original draft. **Chaofeng Li:** Conceptualization, Validation, Supervision, Project administration, Formal analysis, Writing – review & editing. **Qinqin Fan:** Formal analysis, Writing – review & editing. **Yuxin Liu:** Investigation, Writing – review & editing.

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.

Acknowledgments

This work is supported by grants from National Natural Science Foundation of China (No. 62176150) and Shanghai Sailing Program, China (No. 21YF1416700). Thanks to Hideyuki TAKAGI for his valuable comments and discussions on this work.

Appendix. Solutions of variable sizes

Fig. 11 shows instance solutions for TSP with 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 nodes obtained by GCE-MAD NET compared with the optimal solutions found by CPLEX. CPLEX tends to add the nearest nodes next to the current node to the partial route which leads to a higher cost than GCE-MAD NET.

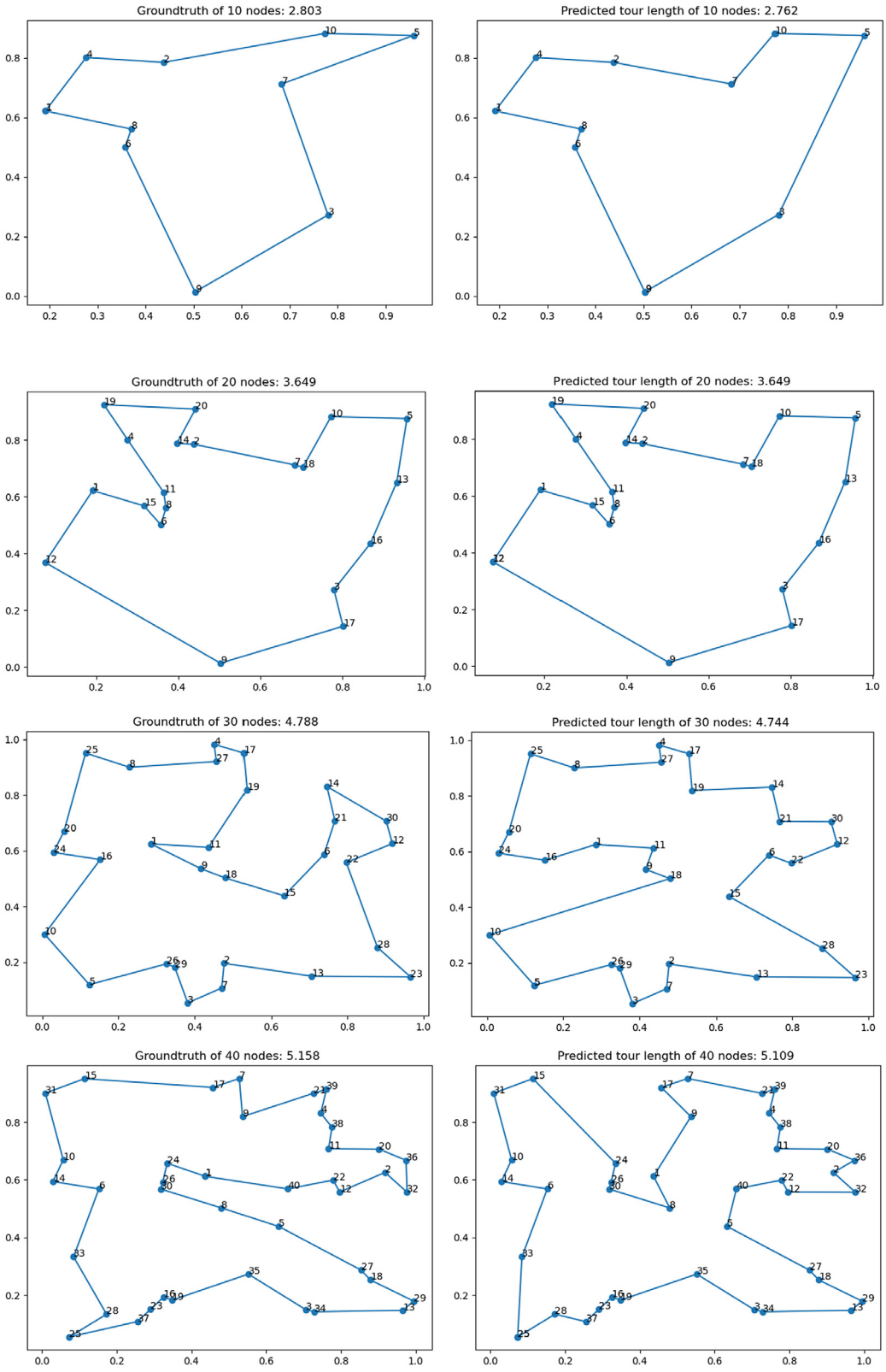


Fig. 11. Random TSP instances of variable sizes are solved by CPLEX (left column) and our model (right column).

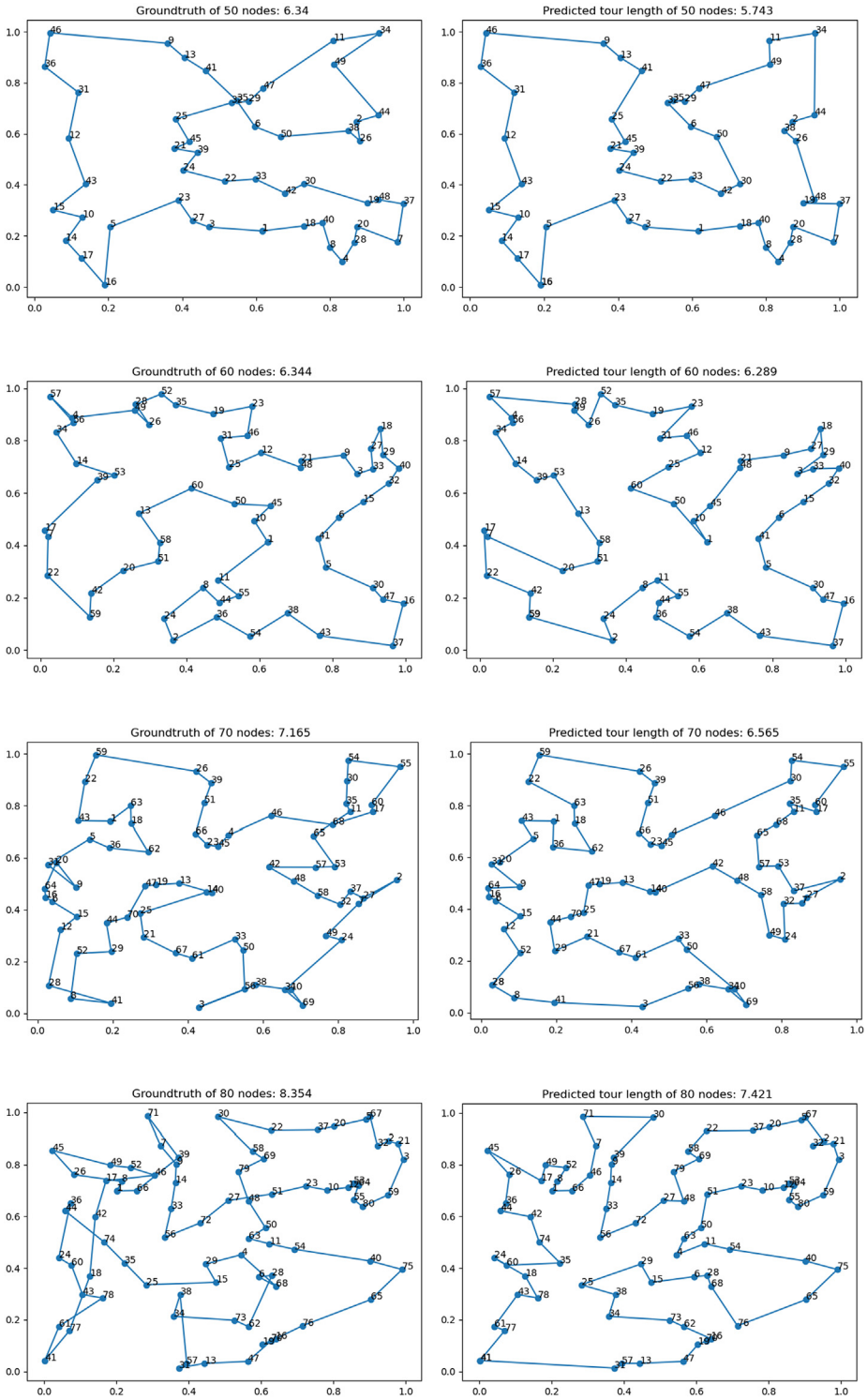


Fig. 11. (*continued*).

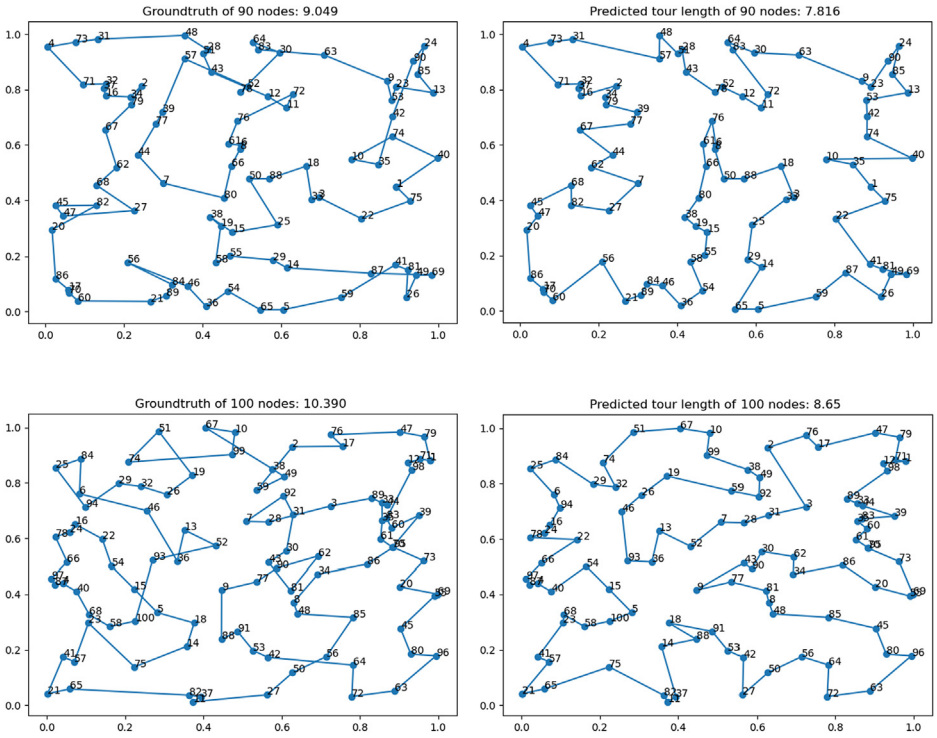


Fig. 11. (*continued*).

References

Al-Gaphari, G.H., Al-Amry, R., Al-Nuzaili, A.S., 2021. Discrete crow-inspired algorithms for traveling salesman problem. Eng. Appl. Artif. Intell. 97, 104006.

Ali, I.M., Essam, D., Kasmarik, K., 2020. A novel design of differential evolution for solving discrete traveling salesman problems. Swarm Evol. Comput. 52, 100607.

Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S., 2016. Neural combinatorial optimization with reinforcement learning. In: International Conference on Learning Representations. San Juan.

Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: a methodological tour d'Horizon. European J. Oper. Res. 290, 405–421.

Bresson, X., Laurent, T., 2017. Residual gated graph ConvNets. arXiv preprint arXiv:1711.07553.

Chen, J., Ma, T., Xiao, C., 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247.

Cho, K., Gulcehre, B.v.M.C., Bahdanau, D., Schwenk, F.B.H., Bengio, Y., 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: EMNLP.

Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L., Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, vol. 30. Long Beach, CA, pp. 6348–6358.

Davendra, D., 2010. Traveling Salesman Problem: Theory and Applications. BoD–Books on Demand.

Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, vol. 29. Barcelona, SPAIN, pp. 3844–3852.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L., 2018. Learning heuristics for the TSP by policy gradient. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Delft, The Netherlands, pp. 170–181.

Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P., 2015. Convolutional networks on graphs for learning molecular fingerprints. In: Advances in Neural Information Processing Systems, vol. 28. pp. 2224–2232.

Ebadinezhad, S., 2020. DEACO: ADopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem. Eng. Appl. Artif. Intell. 92, 103649.

Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N., 2017. Convolutional sequence to sequence learning. In: International Conference on Machine Learning, pp. 1243–1252.

Goyal, P., Ferrara, E., 2018. Graph embedding techniques, applications, and performance: A survey. Knowl.-Based Syst. 151, 78–94.

Hromkovič, J., 2013. Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer Science & Business Media.

Huang, H., Savkin, A.V., Huang, C., 2020. A new parcel delivery system with drones and a public train. J. Intell. Robot. Syst. 100 (3), 31341–31354.

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

Kanna, S.K.R., Sivakumar, K., Lingaraj, N., 2021. Development of deer hunting linked earthworm optimization algorithm for solving large scale traveling salesman problem. Knowl.-Based Syst. 227, 107199.

Khan, I., Maiti, M.K., 2019. A swap sequence based artificial bee colony algorithm for traveling salesman problem. Swarm Evol. Comput. 44, 428–438.

Kinable, J., Smeulders, B., Delcour, E., Spieksma, F.C.R., 2017. Exact algorithms for the equitable traveling salesman problem. European J. Oper. Res. 261 (2), 475–485.

Kool, W., Hoof, H.V., Welling, M., 2018. Attention, learn to solve routing problems! in: International Conference on Learning Representations. Vancouver, BC.

Li, Q., Han, Z., Wu, X.-M., 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In: Thirty-Second AAAI Conference on Artificial Intelligence.

Li, W., Wang, G.-G., Gandomi, A., 2021. A survey of learning-based intelligent optimization algorithms. In: Archives of Computational Methods in Engineering. pp. 1–19.

Marcheggiani, D., Titov, I., 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In: EMNLP.

MirHassani, S.A., Habibi, F., 2013. Solution approaches to the course timetabling problem. 39, (2), pp. 133–149.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Nammouchi, A., Ghazzai, H., Massoud, Y., 2020. A generative graph method to solve the travelling salesman problem. In: IEEE 63rd International Midwest Symposium on Circuits and Systems. pp. 89–92.

Nazari, M., 2018. Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems. pp. 9839–9849.

Osaba, E., Yang, X.-S., Ser, J.Del., 2020. Traveling Salesman Problem: A Perspective Review of Recent Research and New Results with Bio-Inspired Metaheuristics. pp. 135–164.

Pandiri, V., Singh, A., 2019. An artificial bee colony algorithm with variable degree of perturbation for the generalized covering traveling salesman problem. Appl. Soft Comput. 78, 481–495.

Paschos, V.T., 2014. Applications of Combinatorial Optimization, vol. 3. John Wiley & Sons.

Rego, C., Gamboa, D., Glover, F., Osterman, C., 2011. Traveling salesman problem heuristics: Leading methods. Implement. Lat. Adv. 211 (3), 427–441.

Saji, Y., Barkatou, M., 2021. A discrete bat algorithm based on Lévy flights for Euclidean traveling salesman problem. Expert Syst. Appl. 172, 114639.

Subramanyam, A., Gounaris, C.E., 2016. A branch-and-cut framework for the consistent traveling salesman problem. European J. Oper. Res. 248 (2), 384–395.

Talbi, E.-G., 2021. Machine learning into metaheuristics: A survey and taxonomy. ACM Comput. Surv. 54 (6), 1–32.

Tran, D.-D., Vafaeipour, M., Baghdadi, M.El., Barrero, R., Mierlo, J.Van., Hegazy, O., 2020. Thorough state-of-the-art analysis of electric and hybrid vehicle powertrains: Topologies and integrated energy management strategies. Renew. Sustain. Energy Rev. 119, 109596.

Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., Attention is all you need. In: Advances in Neural Information Processing Systems. Long Beach, CA, pp. 5998–6008.

Vinyals, O., Fortunato, M., Jaitly, N., 2015b. Pointer networks. In: Advances in Neural Information Processing Systems, vol. 28. Montréal, Canada, pp. 2692–2700.

Wang, Y., Han, Z., 2021. Ant colony optimization for traveling salesman problem based on parameters optimization. Appl. Soft Comput. 107, 107439.

Wang, Z., Zhang, Y., Zhou, W., Liu, H., 2012. Solving traveling salesman problem in the Adleman–Lipton model. Appl. Math. Comput. 219 (4), 2267–2270.

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

Williamson, D.P., Shmoys, D.B., 2011. The Design of Approximation Algorithms. Cambridge University Press.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y., 2020. A comprehensive survey on graph neural networks. 32, (1), pp. 4–24.

Wu, J.Muren., Zhou, L., Du, Z., Lv, Y., 2019. Mixed steepest descent algorithm for the traveling salesman problem and application in air logistics. Transp. Res. E 126, 87–102.

Zhou, J., 2020. Graph neural networks: A review of methods and applications. AI Open 1, 57–81.