Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 19(10): 9730–9748.
DOI: 10.3934/mbe.2022452
Received: 19 May 2022
Revised: 20 June 2022
Accepted: 23 June 2022
Published: 04 July 2022

Research article

Dynamic graph Conv-LSTM model with dynamic positional encoding for the large-scale traveling salesman problem

Yang Wang and Zhibin Chen*

Department of Mathematics, Kunming University of Science and Technology, Kunming 650093, China

* **Correspondence:** Email: chenzhibin311@126.com.

Abstract: Recent research has showen that deep reinforcement learning (DRL) can be used to design better heuristics for the traveling salesman problem (TSP) on the small scale, but does not do well when generalized to large instances. In order to improve the generalization ability of the model when the nodes change from small to large, we propose a dynamic graph Conv-LSTM model (DGCM) to the solve large-scale TSP. The noted feature of our model is the use of a dynamic encoder-decoder architecture and a convolution long short-term memory network, which enable the model to capture the topological structure of the graph dynamically, as well as the potential relationships between nodes. In addition, we propose a dynamic positional encoding layer in the DGCM, which can improve the quality of solutions by providing more location information. The experimental results show that the performance of the DGCM on the large-scale TSP surpasses the state-of-the-art DRL-based methods and yields good performance when generalized to real-world datasets. Moreover, our model compares favorably to heuristic algorithms and professional solvers in terms of computational time.

Keywords: dynamic graph Conv-LSTM model; traveling salesman problem; deep reinforcement learning; dynamic positional encoding; learning heuristics

1. Introduction

The traveling salesman problem (TSP) is a well-known combinatorial optimization problem (COP), whereby the given are n cities and $(n-1)n/2$ non-negative integers denoting the distances between all pairs of cities. The objective of a TSP is to find a closed tour with the shortest length that visits all cities only once and returns to the origin city [1]. The TSP is an NP-hard problem [2], even in the symmetric two-dimensional Euclidean version, which is this work's focus. Even though the TSP is very undesirable because of the computational time, many heuristics and exact algorithms are known to handle the problem. So for some instances of tens of thousands of cities, we can solve for the TSP approximation; even for the problems with millions of cities, we can approximate it within a small fraction of 1% [3]. Nevertheless, there is no perfect strategy for solving the TSP completely. Exact algorithms [4], such as branch-and-bound and dynamic programming, have the theoretical guarantee of finding the optimal solutions, but computational complexity increases exponential with increases in the number of nodes. Approximation algorithms [5], such as those based on local search and linear programming, can quickly yield near-optimal solutions within polynomial time, but they may only apply to specific problems. In practice, heuristic algorithms [6], such as ant colony optimization and particle swarm optimization, are the most commonly applied approaches for solving TSP within an acceptable running time, especially for large-scale TSP. However, designing heuristics is not straightforward as it requires a lot of trials. The quality of TSP solutions is highly dependent on one's knowledge of problem-specific expert and hand-crafted features.

The practice of applying machine learning to solve TSP has a long history. For example, as far back as the 1980s, the Hopfield neural network [7] has been used to solve TSP. Recently, there has been a growing trend toward applying deep reinforcement learning (DRL) and graph neural networks (GNNs) to automatically discover faster heuristic algorithms to solve TSP [8, 9]. Instead of applying experts to manually design heuristics and rules, neural networks learn these heuristics and rules by imitating the best solver or by DRL. Compared to manual algorithm designs, people believe it is feasible to apply DRL and GNNs in the decision-making or heuristic algorithms to solve TSP. Even though heuristic algorithms operate well on TSP, once the problem statement changes slightly, they need to be revised. In contrast, DRL-based methods have the potential to find useful features that may be hard to specify by human algorithm designers, allowing it to develop a better solution [10]. The model policies can be parameterized by neural networks and trained by DRL to obtain more robust algorithms for TSP.

Once the training is completed, the model can directly output the solution of the TSP, yielding a faster solution. Most of these DRL-based methods follow the encoder-decoder structure and learning construction heuristics, their architecture can be roughly classified as either: (1) an autoregressive model, which builds the solution set step by step, or (2) a non-autoregressive model, wherein all solutions are generated at one time. Particularly, the encoder maps the information of the nodes into feature representation; the decoder then generates an output that predicts the probability of selecting the next node at every construction step. To improve solution quality, DRL-based methods need to be used in combination with some traditional algorithms in the inference phase, algorithms such as greedy, beam search, 2-opt and sampling [11]. However, scale is still an issue for the current DRL-based methods, and generalization ability will be affected when there is a big difference in data distribution between test and training instances.

In this paper, motivated by the recent progress of graph pointer network (GPN) architecture for solving the large-scale TSP [12], we propose a novel policy approach that can achieve excellent performance with generalization ability for a reasonable computational cost. The contributions of this work are three-fold. First, we propose a dynamic graph convolutional long short-term memory (hereafter referred to as Conv-LSTM) model (DGCM) with dynamic encoder-decoder architecture to train construction policies at different construction steps. It employs GNNs and a Conv-LSTM network to encode node information and an attention mechanism (AM) as a decoder. Second, to improve the solution quality of trained construction policies, we propose a dynamic positional encoding (DPE) layer in the DGCM which is used in the decoder structure. It can make the nodes satisfy translation invariance during the embedding process. Further, the decoder stage will get more location information to facilitate selection of the next node, thereby increasing the quality of the solution. Finally, we empirically show that the DGCM is a state-of-the-art DRL-based method solver for the large-scale TSP, and that it also demonstrates superior performance to heuristic algorithms and professional solvers in a short inference time. The experimental results concretely show that the DGCM performs significantly better than GPN and obviously decreases the optimality gap.

The remainder of this paper is structured as follows. Section 2 gives a brief overview of recent relevant work. Sections 3 and 4 describe the DGCM model for TSP and the training method, respectively. The experimental results are given in Section 5. Section 6 concludes this paper.

2. Literature review

When using heuristic algorithms for TSP, the entire distance matrix must be recalculated and the system must be re-optimized from scratch, which is impractical, especially for the large-scale TSP. In contrast, the DRL framework does not require an explicit distance matrix, and only one feed-forward pass of the network will update the routes based on the new data generated by environmental interactions. Vinyals et al. [13] improved the sequence-to-sequence (Seq2Seq) model [14] and proposed a pointer network with an long short-term memory (LSTM) network as the encoder and an AM as the decoder. It can effectively solve small-scale TSP. Bello et al. [15] used reinforcement learning (RL) based on a pointer network to solve TSP within 100 nodes.

GNNs, as a powerful tool, can effectively handle non-Euclidean data. Applying GNNs as their basis, Dai et al. [16] proposed a graph embedding network for large-scale COP. The network parameters are trained by a deep Q-learning algorithm that solves the large-scale TSP. Partial solutions are embedded as graphs, and the deep neural network estimates the value of each graph. Joshi et al. [17] proposed an efficient graph convolution network technique for TSP. This method ignores the sequential nature of TSP, and the training efficiency may be low. Ma et al. [12] introduced a GPN model that is trained using RL for tackling large-scale TSP. The GPN generalizes well from small-scale to larger-scale TSP, but the DGCM outperforms the GPN on both small and larger TSP instances.

The transformer is different from the previous structure, which does not require recursion but is entirely dependent on the AM to describe the global dependency between input and output. Applying the transformer as the basis, Kool et al. [18] applied an AM model to solve TSP for the first time. Specifically, the TSP is converted to the Seq2Seq model, explicitly simulating the sequential induction bias of TSP by selecting one node at a time. Wu et al. [19] proposed a direct policy approach that parameterizes the policy model by using the self-attention mechanism to obtain the solution of the TSP in the model training phase. An immediate shortcoming of the AM model is that it does not take into account the underlying symmetry of TSP. Xin et al. [20] proposed a multi-decoder attention model (MDAM) model to solve the multi-objective routing problems and added an embedding glimpse which improves the overall optimization performance of the model in the encoding. Kwon et al. [21] proposed a policy optimization with multiple optima (POMO) framework that entails the use of DRL to train multiple initial nodes of multi-head attention (MHA) models to solve different problems, including TSP. Further, to address the drawback of incomplete node information that is associated with the traditional positional encoding technique, Ma et al. [22] proposed a dual-aspect collaborative transformer (DACT) model and a cyclic positional encoding method to solve TSP with dynamically and cyclically changing nodes. For the initial solution instability problem, Kool et al.

[23] integrated DRL, a transformer and dynamic programming and proposed a deep policy dynamic programming (DPDP) model to solve TSP. Bresson and Laurent [24] use the same transformer encoder but the decoder architecture is different. It constructs the query using all cities in a partial tour with a self-attention module. Hudson et al. [25] proposed a hybrid data-driven approach for solving TSP based on GNNs and guided local searches. This method enhances the generalization ability of the model. Xin et al. [26] proposed a new algorithm NeuroLKH that combines deep learning with the strong conventional heuristic Lin-Kernighan-Helsgaun (LKH) to solve TSP. Traditional methods combined with RL enhance the performance of the LKH algorithm.

Those models can learn to choose appropriate solutions for a TSP from the vast potential solutions of the combinatorial space. Therefore, in view of the poor generalization ability of the model and the large deficit of practical solutions for the large-scale TSP, we propose the DGCM model to solve the large-scale TSP efficiently and provide an effective strategy for solving COP.

3. Deep reinforcement model for TSP

The selection of nodes emphasizes environmental factors ; it is naturally similar to the behavior selection of DRL that will affect the decision. The DGCM model consists of an encoder, decoder and training, as shown in Figure 1. We combine GNNs, a Conv-LSTM network and search strategies to make it easier for the model to a handle large-scale TSP with up to 1000 nodes. DRL is often an elegant alternative to a poorly researched problem in the absence of a standard solution. Since TSP typically require sequential decisions to minimize problem specific cost functions, they can be elegantly fed into the DRL framework, which trains agents to maximize the reward function (the negative value of the loss function). Hence, DRL algorithms are used to train the parameters θ and input instance s; the probability of the solution $p_\theta(a_t \mid s)$ can be decomposed by the chain rule as

$$p_\theta(a_t \mid s) = \prod_{t=1}^{N} p_\theta(a_t \mid s, a_{1:t-1}). \tag{3.1}$$

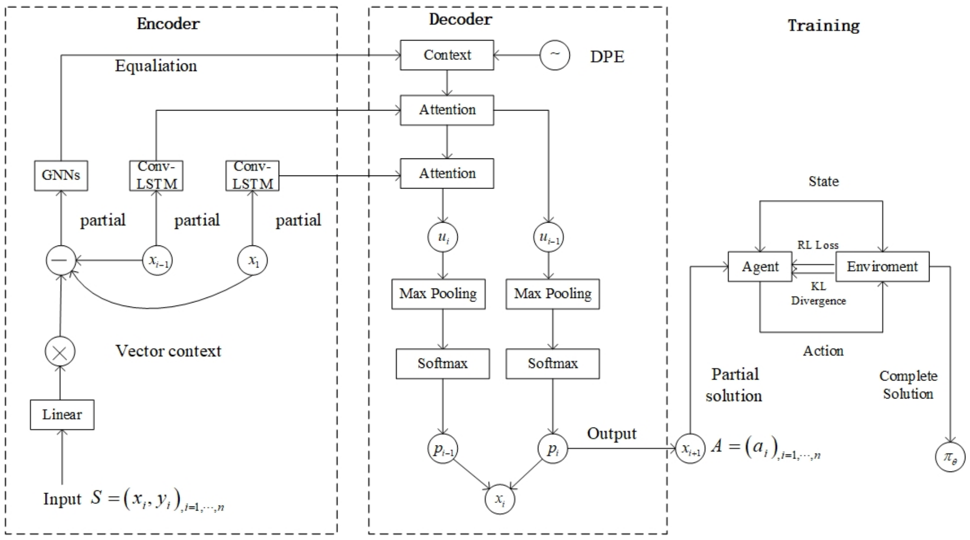

Figure 1. Diagram of DGCM frame.

3.1. GNNs and Conv-LSTM based encoder

In this work, similar to [12], we also use the vectors pointing from the current node to all other nodes as the embedding, which we refer to as the vector context. The vector context is constructed incrementally by the embedding. At different construction steps, the state of the instance is changed, and the feature embedding of each node should be updated. For the current node x_i, suppose $X_i = [X_1^T, \ldots, X_i^T] \in R^{N\times 2}$ is a matrix with identical rows x_i. We define $\dot{X}_i = X - X_i$ as the vector context.

The encoder includes a node encoder derived from the Conv-LSTM and a graph encoder based on GNNs. For the node encoder, each current node coordinate x_i is embedded in a higher dimensional vector $\tilde{x}_i \in R^d$, where d is the hidden dimension. Since the LSTM networks in previous works did not take spatial correlation into account and contained a large amount of redundant spatial data , we use a Conv-LSTM network to extract the spatial and temporal characteristics in the encoding structure. The core essence of Conv-LSTM is still the same as LSTM, the output of the previous layer is used as the input of the next layer. The difference is that with the addition of the convolution operation, not only can we obtain the temporal relationships, but we can also extract the spatial features like the convolution layer. In this way, the spatio-temporal characteristics can be obtained, and the calculation for switching between states can be replaced by convolution. We follow the Conv-LSTM formula in [27]. This is expressed as follows

$$i_t = \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \odot C_{t-1} + b_i), \tag{3.2}$$

$$f_t = \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{ci} \odot C_{t-1} + b_f), \tag{3.3}$$

$$C_t = f_t \odot C_{t-1} + i_t + \tan\odot(W_{xc} * x_t + W_{hc} * C_{t-1} + b_i), \tag{3.4}$$

$$o_t = \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \odot C_{t-1} + b_o), \tag{3.5}$$

$$h_t = o_t \odot (C_t), \tag{3.6}$$

where i_t, f_t and o_t respectively denote the input gate, forget gate and output gate. Here i_t, f_t and o_t are all three-dimensional tensors, and their last two dimensions respectively represent the spatial information of rows and columns (the first dimension is the temporal dimension). x_t represents a one-dimensional vector or scalar, and h_t can be given a different dimension. C_t determines how much information this output takes from this input and is leftover from the last one. The weighted parameter metrices are $W_{xi} \sim W_{co}$, which conduct a linear transformation between the vectors. $b_i \sim b_o$ are the intercept parameters. The operator $\odot$ is the Hadamard product, and $*$ denotes convolution.

For the graph encoder, each node is related to its neighbor nodes, and they will abstract as an issue between the set of nodes and edges. We use graph embedding (GE) layers to encode all node coordinates $x = [x_1^T, \ldots, x_N^T]^T$. Therefore, all feature vectors can be derived from the aggregation operation of GNNs in GE layers. In this way, the network can effectively capture the topological structure of the graph and the potential relationship between nodes, so that more information can be represented. And encoder information embedding will have better performance. The expression of the GE structure is described as

$$x^l = \gamma x^{l-1}\phi + (1-\gamma)\psi_\theta(\frac{x^{l-1}}{|N(i)|}), \tag{3.7}$$

where $x^l \in R^{N\times d_l}$, γ is a trainable parameter that adjusts the weight matrix of eigenvalues, $\phi \in R^{N\times d_l}$, $\psi_\theta : R^{N\times d_l - 1} \to R^{N\times d_l}$ is the aggregation function [28] and $N(i)$ is the adjacency set of Node i.

GNNs essentially reduce the search space of the search algorithm. The key to the strong generalization ability of the model is to successfully transfer the learning strategy to a larger graph so that the prediction results of the encoder can still have generalization ability when the TSP changes from small to large. We consider a TSP that is characterized by a complete graph with symmetry. In order to maintain the global properties of the decoding structure and weight distribution of attention, x^l is homogenized which makes the aggregated attribute information uniformly embedded in the context vector. Here the expression of the GE structure is described as

$$\bar{X} = \sum_{l=1}^{L} x^l. \tag{3.8}$$

3.2. Attention and DPE-based decoder

Compared with the previous method, this dynamic construction method can be similar to the idea of the divide-and-conquer method. The problem is decomposed into several sub-problems to learn the hierarchical strategy, and then the strategies of the sub-problems are combined to form the global optimal strategy, which implicitly leads to a better generalization effect for the larger problem instances. The structure of this new solution is different from the original solution, and decoding based on the same embedding for all the construction steps may lead to poor performance. To address it, we propose the DPE technique to learn the effective location information in the decoder structure. In the process of embedding, the node coordinates can satisfy translation invariance. Meanwhile, the high-level neural network can extract more task-related features. Here, the DPE of each node i is defined as

$$DPE_{t,i} = \begin{cases} \sin(2\pi f_i t + \frac{2\pi}{\omega_d i}), i \text{ is odd} \\ \cos(2\pi f_i t + \frac{2\pi}{\omega_d i}), i \text{ is even} \end{cases} \tag{3.9}$$

where

$$f_i = \frac{10000^{\frac{d}{2i}}}{2\pi} \tag{3.10}$$

$$\omega_d = \begin{cases} \frac{3\lfloor d/3 \rfloor + 1}{d}(N - N^{\frac{1}{\lfloor d/2 \rfloor}}) + \frac{1}{\lfloor d/2 \rfloor}, \text{if } d < \lfloor \frac{d}{2} \rfloor \\ N \qquad , \text{otherwise} \end{cases} \tag{3.11}$$

$DPE_{t,i} \in R^d$, t is the location of the node and $d = 128$ is the embedding dimension; $i \in \{1, 2, \ldots, n\}$ and the angular frequency ω_d is decreasing along the dimension to make the wavelength longer within the range N [22].

Similar to [18], the vector context is computed by an AM and outputs the pointer vector u_i. Masking technology that ensures that the visited nodes cannot be accessed again can be understood as the output of the next visited city node with a high probability. The AM and pointer vector u_i are defined as

$$u_i^{(j)} = \begin{cases} C \cdot \tanh(W_r r_j + W_q q), \text{if } j \neq \pi_{t'}\ \forall t' < t \\ -\infty \qquad , \text{otherwise} \end{cases} \tag{3.12}$$

where $u_i^{(j)}$ is the *j-th* entry of the vector u_i, W_r and W_q are trainable matrices, q is a query vector from the hidden variable of the Conv-LSTM algorithm and r_j is a reference vector containing the information on the context of all cities.

Given the node embedding u_i by the attention decoder, we aggregate them by max-pooling to get GE. In this way, the global graph information of a solution is effectively fused into its nodes. Figure 2 shows that the dynamic encoder-decoder architecture and inference can be applied to solve TSP. The distribution policy overall candidate nodes are given by

$$\pi_\theta(a_i \mid s_i) = p_i = \text{softmax}(\max(u_i)). \tag{3.13}$$

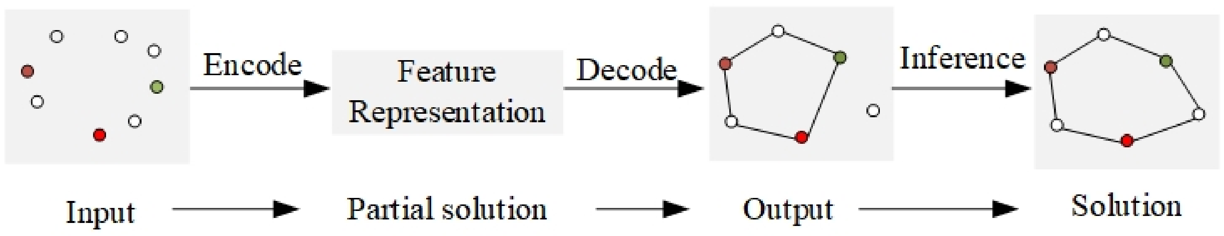

Figure 2. Diagram of dynamic encoder-decoder architecture for TSPs.

3.3. DRL training with multiple loss optimization

The entire encoder-decoder architecture is trained in an end-to-end manner, and the model can be trained through DRL to produce near-optimal solutions. In order to measure the difference distribution between partial solutions and the greedy policy π_θ^g, we add Kullback-Leibler (KL) divergence loss to the baseline of the REINFORCE algorithm [29]. By using KL divergence, we can bring the greedy policy π_θ^g infinitely closer to the real solution, so that the prediction is more accurate. Here KL divergence loss is expressed as

$$D_{KL} = \sum_{i=1}^{N} \pi_\theta(a_i \mid s_i) \log(\pi_\theta(a_i \mid s_i)/\pi_\theta(a_i \mid s_i)). \tag{3.14}$$

During training, routing is drawn from a distribution s and the total training objective is defined as

$$J_\theta = E_{\pi \sim p_\theta}(J(\theta \mid s)). \tag{3.15}$$

We sample solution trajectories a_i^s and adopt a policy gradient to find a parameter θ. In order to maximize the expected return J and circumvent non-differentiability of hard-attention, the DGCM has recourse to the well-known REINFORCE algorithm [29] learning rule. The gradient of $J(\theta)$ can be expressed by

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (R(a_i^s \mid s_i) - R(a_i^g \mid s_i)) \nabla_\theta \log p_\theta(a_i \mid s_i)). \tag{3.16}$$

The model can learn the parameter θ of the actor network through a random strategy. The gradient of the above formula is calculated and updated; then, the optimal strategy P_i is obtained through iterative training. We use a dual loss control model for convergence and training rewards. The update rule can be expressed by

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) + \beta \nabla_\theta D_{KL}(\theta), \tag{3.17}$$

where α is the learning rate and β is the coefficient of KL loss.

The training algorithm is described in Algorithm 1. The algorithm terminates once a pre-defined maximum number of iterations is reached.

Algorithm 1 Multiple loss optimization algorithm

```
Require: a differentiable policy parameterization $\pi_\theta(a \mid s, \theta)$, number of epochs $B$, steps per epoch $T$,
    training set $S$
Ensure: policy $\pi_\theta$
 1: initialize policy network parameter $\theta$,$\phi$
 2: repeat epoch=1,. . .,B do
 3:     for step=1,. . .,T do
 4:     $s_i \leftarrow$ SampleInput(S) for $i \in 1, \ldots, B$
 5:     $R(a_i^s \mid s_i) \leftarrow$ SampleRollout($s_i$) for $i \in 1, \ldots, B$
 6:     $R(a_i^g \mid s_i) \leftarrow$ GreedyRollout($s_i$) for $i \in 1, \ldots, B$
 7:     $\nabla_\theta J(\theta) \leftarrow \frac{1}{N} \sum_{i=1}^{N} (R(a_i^s \mid s_i) - R(a_i^g \mid s_i)) \nabla_\theta \log p_\theta(a_i \mid s_i))$
 8:     Compute $\nabla_\theta D_{KL}(\theta)$
 9:     $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) + \beta \nabla_\theta D_{KL}(\theta)$
10:     end for
11: until a pre-defined maximum number of iterations is reached
```

4. Numerical studies

4.1. Experimental environment and hyperparameters

In each epoch, the training data are generated randomly in the unit square $[0, 1]\times[0, 1]$; the instances used in our experiments were symmetrical TSP with 20, 50 and 100 nodes, respectively. We call them TSP20, TSP50 and TSP100, respectively. The DGCM model is executed on a single GPU Tesla K80. Note that 100 epochs require, on average 3, 12 and 36 h for TSP20, TSP50 and TSP100, respectively. We trained with TSP50 instances and generalize to large-scale TSP. In the testing phase, we adopted greedy and 2-opt inferencing methods to improve solution quality. We report the objective value, gap and runtime metrics of the TSP separately. To ensure the fairness of the experiment, we set the same hyperparameters based on [12]. Due to GPU memory limitations, we set the training batch size to a uniform 256. The experiments entailed the use of the hyperparameters shown in Table 1.

4.2. Parameter tuning

The relevant parameters of the DRL model are critical to the quality of the solution. The optimal values after the optimization will be used in the numerical experiments that followed. In this study, the GE operation aggregation point feature was used to test the effects of different layers of GE operation on the model performance on TSP with 20, 50 or 100 nodes. After several tunings, as shown in Table 2,

Table 1. Hyperparameters used for training.

Parameter	Value	Parameter	Value
Epoch	100	Optimizer	Adma
Batch size	256	Learning rate	10^{-3}
GE layers	3	Learning rate decay	0.96
Training instances	2500	Test instances	1000

the parameters with the optimal target values were selected. We found that, as the number of GE layers increases, the inference time increases accordingly due to the increase in network parameters. However, the GE operation becomes progressively less effective above three layers. The reasonableness of the parameter settings for these experiments have also been indirectly verified.

Table 2. GE layer parameter tuning.

	TSP20			TSP50			TSP100		
Parameter	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
GE layers (1)	3.85	0.52%	0.5 *s*	5.73	0.70%	2 *s*	7.82	0.77%	13 *s*
GE layers (2)	3.84	0.39%	0.8 *s*	5.71	0.43%	2 *s*	7.79	0.45%	15 *s*
GE layers (3)	**3.83**	0.00%	1 *s*	**5.70**	0.17%	3 *s*	**7.79**	0.38%	16 *s*
GE layers (4)	3.84	0.26%	2 *s*	5.72	0.62%	5 *s*	7.80	0.51%	20 *s*
GE layers (5)	3.84	0.41%	3 *s*	5.72	0.67%	6 *s*	7.81	0.71%	23 *s*

Note: bold is the best result and the precision is 10^{-3}. The Obj is the objective value, same as below.

4.3. *Experiments for small-scale TSP*

Professional solving tools, such as the Concorde [30] and LKH algorithms[31], were used to calculate the optimal solution of the TSP. Concorde and LKH algorithms were run on an Intel Core i5-9300H CPU. In Table 3, we compare the performance of our model on small-scale TSP with other baselines. The baseline models include professional solving tools, traditional algorithms and DRL-based methods. Traditional solvers like Gurobi [32] and OR-Tools [33] still outperform DRL-based solvers in terms of performance and generalization. However, they can only provide weaker solutions or would take very long to solve the TSP. Although it took 3 h in the training phase, the DGCM only takes 1 s in the inferencing phase to get the suboptimal solution of TSP20. Once the model training is completed, it can be generalized to solve large-scale TSP. For the optimal gap of TSP50/100, our model is inferior to POMO [21], DACT [22] and DPDP [23] but surpasses other current DRL-based models. In all small-scale TSP distances, the overall performance of our model is superior to traditional algorithms except for professional solvers. The 2-opt inference method resulted in an optimal gap of 0.21% for TSP20, 0.51% for TSP50 and 0.76% for TSP100; it is shown that the combined use of the DGCM and 2-opt method can reduce the optimal gap even further. Compared with the GPN model [12], the performance of the DGCM was notably improved for TSP20 (6.14%), TSP50 (5.96%) and TSP100 (13.74%).

Table 3. Experiments for small-scale TSP.

	TSP20			TSP50			TSP100		
Model	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
Concorde	3.83	0.00%	5 *min*	5.69	0.00%	13 *min*	7.76	0.00%	1 *h*
Gurobi*	3.83	0.00%	7 *s*	5.69	0.00%	2 *min*	7.76	0.00%	17 *min*
OR-Tools*	3.86	0.94%	1 *min*	5.85	2.87%	5 *min*	8.06	3.86%	23 *min*
LKH	3.83	0.00%	42 *s*	5.69	0.00%	6 *min*	7.76	0.00%	25 *min*
2-opt	3.95	3.31%	1*s*	6.11	7.38%	7 *s*	8.50	9.53%	33 *s*
Farthest insertion	3.89	1.56%	1 *s*	5.97	4.92%	2 *s*	8.34	7.47%	10 *s*
Nearest neighbor	4.48	16.9%	1 *s*	6.94	21.9%	3*s*	9.68	24.7%	7*s*
AM (greedy)*	3.85	0.34%	1 *s*	6.94	5.80%	2*s*	8.12	4.53%	6 *s*
AM (sampling)*	3.84	0.08%	5 *min*	5.80	5.73%	24 *min*	7.94	2.26%	1 *h*
Coast*	3.83	0.00%	15 *min*	5.71	0.35%	29 *min*	7.83	0.87%	41 *min*
Wu*	3.83	0.00%	1 *min*	5.70	0.26%	1.5 *h*	7.87	1.42%	2 *h*
POMO	3.83	3.42%	1 *s*	**5.69**	0.08%	16 *s*	7.78	0.26%	1 *min*
DACT	3.83	0.00%	10 *min*	5.70	0.18%	1 *h*	7.77	0.19%	2.5 *h*
MDAM (BS)	3.83	0.00%	3 *min*	5.70	0.18%	14 *min*	7.79	0.39%	44 *min*
DPDP*	—	—	—	—	—	—	**7.77**	0.13%	3 *h*
Hudson*	3.83	0.00%	10 *s*	5.69	0.07%	10 *s*	7.81	0.69%	10 *s*
GPN	3.89	1.61%	1 *s*	6.03	5.97%	3 *s*	8.87	14.3%	6 *s*
DGCM (greedy)	3.89	1.56%	1 *s*	5.99	5.25%	2*s*	8.63	11.2%	8*s*
DGCM (2-opt)	**3.83**	0.00%	1 *s*	5.70	0.17%	3 *s*	7.79	0.38%	16 *s*

Note: bold is the best among learning based methods. Results with * are reported from others papers. The precision is 10^{-3}.

4.4. Experiments for large-scale TSP

Regarding the results in Table 4, the current DRL-based methods have poor generalization ability and give worse results than heuristics. The generalization ability of the DGCM was better by an order of magnitude. We observe that for TSP250/500/750/1000, our model surpasses the current DRL-based models on path length. The advantage of our model is shorter time as compared with traditional algorithms, which allows it to perform significantly better than DRL-based methods and obviously decreases the routing cost. The running time of the large-scale TSP was also shortened, compared with some traditional algorithms. In terms of the tradeoff between time and routing cost, the DGCM performs better than GPN and other baseline methods. The DGCM with dynamic encoder-decoder architecture can explore structural features dynamically and exploit hidden structure information effectively at different construction steps. The key of the DGCM is to distribute the computational solutions in different construction steps, while the DPE takes into account the hidden and dynamic node structure information. Hence, more structured information can be represented and lead to better performance. To directly express the superiority of the result, we take the LKH algorithm as the benchmark. Even though our model is not as effective in terms of optimization, it demonstrated good generalization performance and still has the potential to be an effective solution method. Compared with the GPN model [12], the performance of the DGCM was notably improved for TSP250 (8.37%), TSP500 (7.19%), TSP750 (7.81%) and TSP1000 (8.55%).

Table 4. Experiments for large-scale TSP.

	TSP250		TSP500		TSP750		TSP1000	
Model	Obj	Time	Obj	Time	Obj	Time	Obj	Time
LKH	11.893	9792 *s*	16.524	23070 *s*	20.129	36840 *s*	23.130	50680 *s*
Concorde	11.89	1894 *s*	16.55	13902 *s*	20.10	32993 *s*	23.11	47804 *s*
OR Tools*	12.289	5000 *s*	17.449	5000 *s*	22.395	5000 *s*	26.477	5000 *s*
Nearest Insertion	14.928	25 *s*	20.791	60 s	25.291	115 s	28.973	136 s
2-opt	13.026	303 *s*	18.600	1363 *s*	22.668	3296 *s*	26.111	6153 *s*
Farthest Insertion	13.026	33 *s*	18.288	160 *s*	22.342	454 *s*	25.741	945 *s*
PN*	14.249	29 *s*	21.409	280 *s*	27.382	782 *s*	32.714	3133 *s*
S2V-DQN*	13.097	476 *s*	18.428	1508 *s*	22.550	3182 *s*	26.046	5600 *s*
AM*	14.032	2 *s*	24.789	14 *s*	28.281	42 *s*	34.055	136 *s*
GPN (Greedy)*	13.765	32 *s*	19.829	111 *s*	24.679	232 *s*	28.929	393 *s*
GPN (2-opt)*	12.971	214 *s*	18.361	974 *s*	22.519	2278 *s*	26.013	4410 *s*
DGCM (Greedy)	13.348	29 *s*	19.206	98 *s*	23.980	184 *s*	28.138	305 *s*
DGCM (2-opt)	**12.639**	198 *s*	**17.898**	847 *s*	**21.956**	1986 *s*	**25.400**	3964 *s*

Note: bold is the best among learning based methods. Results with * are reported from others papers. The precision is 10^{-3}.

4.5. Comparison of model generalization ability and convergence

More specifically, we trained the DGCM on TSP20/50/100 and used their models to predict on TSP250/500/750/1000. Once the training was completed, the DGCM directly output the solution of the TSP, confirming faster solving capability. The DGCM can generalize and solve any similarly sized problem. The comparison results for the generalization ability of TSP are shown in Table 5. The results of numerical experiments show that our model can achieve excellent performance with generalization ability for a reasonable computational cost, and that the results will improve if we increase the size of the TSP instances used for training. The mutual generalization ability of small-scale and large-scale TSP indicated good performance.

Table 5. Comparison of generalization ability on TSP.

	TSP250		TSP500		TSP750		TSP1000	
Model	Obj	Time	Obj	Time	Obj	Time	Obj	Time
Ours (TSP20)	14.764	42 *s*	22.201	121 *s*	28.431	223 *s*	34.072	385 *s*
Ours (TSP50)	13.369	29 *s*	19.240	98 *s*	23.980	184 *s*	28.183	305 *s*
Ours (TSP100)	12.906	25 *s*	18.807	84 *s*	23.118	169 *s*	26.806	284 *s*

The convergence comparison between TSP20 and TSP50 given by Figures 3 and 4 show that our model can converge stably within 200 batches. For TSP20, the final training effect of the double loss algorithm is slightly better. For TSP50, the training effect of the multiple loss algorithm was obviously superior; as compared with the single REINFORCE algorithm and GPN model, the multiple loss algorithm exhibited a faster convergence rate and better convergence effect during the training process.

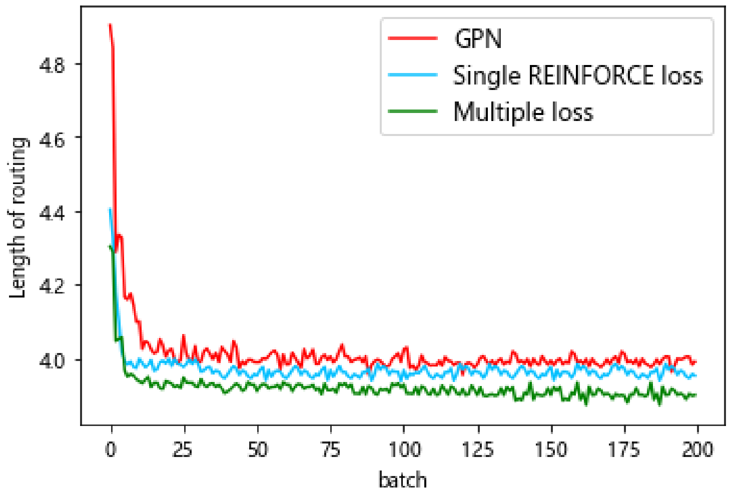

Figure 3. Convergence of TSP20.

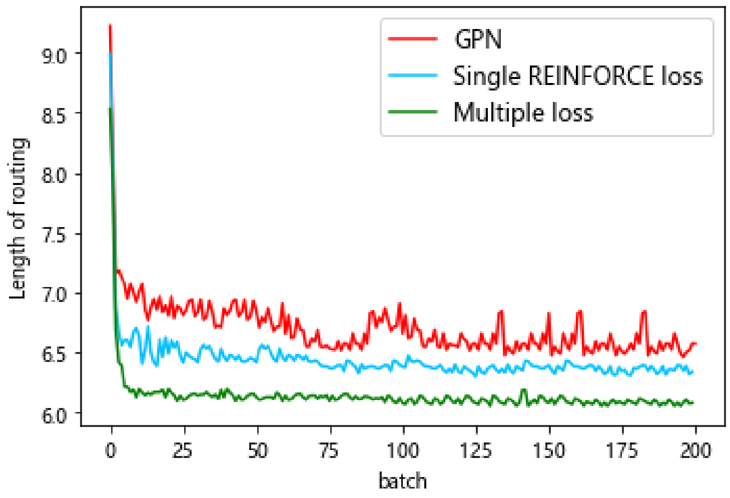

Figure 4. Convergence of TSP50.

4.6. Ablation experiments

DPE is used for cyclic encoding dynamic sequencing so that the initial node coordinates can satisfy the translation invariance during the process of embedding. To demonstrate the importance of the DPE technique, we further evaluated the effectiveness of the DGCM. For the ablation study on small-scale and large-scale TSP, we have excluded the 2-opt inference technique here since it is an independent technique applicable to the inferencing phase. The results are summarized in Tables 6 and 7. We can observe that both the DGCM and DPE consistently improve the quality of the learned construction policy for dynamic encoder-decoder architecture. The results of numerical experiments also show that DPE has little effect on inferencing time.

Table 6. DGPN structure ablation results for small-scale TSP.

	TSP20			TSP50			TSP100		
Model	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
GPN (not DPE-greedy)	4.07	6.35%	1 *s*	6.06	6.47%	3 s	8.89	14.5%	6 *s*
GPN (DPE Greedy)	3.89	1.61%	1 *s*	6.03	5.97%	3 *s*	8.87	14.3%	6 *s*
DGCM (not DPE-greedy)	3.90	1.82%	1 *s*	6.03	5.97%	2 *s*	8.65	11.5%	7 *s*
DGCM (DPE Greedy)	3.88	1.31%	1 *s*	5.98	5.09%	2 *s*	8.52	9.79%	8 *s*

Table 7. DGPN structure ablation results for large-scale TSP.

	TSP250		TSP500		TSP750		TSP1000	
Model	Obj	Time	Obj	Time	Obj	Time	Obj	Time
GPN (not DPE-greedy)	13.784	29 *s*	19.850	101 *s*	24.704	208 *s*	28.996	339 *s*
GPN (DPE Greedy)	13.695	33 *s*	19.742	105 *s*	24.599	212 *s*	28.871	346 *s*
DGCM (not DPE-greedy)	13.365	25 *s*	19.240	89 *s*	23.992	168 *s*	28.149	281 *s*
DGCM (DPE Greedy)	13.348	27 *s*	19.206	85 *s*	23.980	184 *s*	28.138	305 *s*

4.7. Test on real-world datasets

In a real-world application, most real-world instances of TSP would have hundreds or thousands of nodes, and the optimal solution would not be computationally efficient. We further verified that our model can use synthetic data for training and performs fairly well on an instance of the public benchmark TSPlib [34], which contains examples of real-world problems.

We report the results on 30 instances with sizes between 50 and 200 for TSPlib. In Table 8, we find that our model generalizes well from the training model to the real-world dataset , and that it reduced the overall average gap to 0.76%. Though trained on uniform distribution, our model outperforms, GPN [12], AM [18], Wu et al. [19], POMO [21], DACT [22], Hudson et al. [25] and OR-Tools [33] algorithms in terms of the gap in all instances. Given the advantages of our model, it was able to achieve the new state-of-the-art generalization performance among existing DRL-based models on TSPlib benchmark instances with various sizes and distributions. Finally, to ensure the effectiveness of the experiment, we visualized the construction of four tours using DGCM+2-opt. The tours of eil51, KroA100, ch130 and KroB200 are shown in Figures 5–8, respectively. The DGCM model selected each node in the instance more accurately and constructed a shorter route than the the GPN model, which indirectly reflects the effectiveness of the model in this paper.

Table 8. Generalization of DGCM on TSPlib benchmark dataset.

Instance	Opt.	OR-Tools*	Ma et al.	AM*	Wu et al.*	POMO*	DACT*	Hudson et al.*	Ours
eil51	426	436	435	436	438	**426**	**426**	**426**	427
berlin52	7542	7945	7655	7717	8020	**7544**	**7544**	7552	7550
st70	675	683	695	691	706	677	677	680	**675**
eil76	538	561	556	564	575	546	550	**539**	540
pr76	108,159	111104	110654	111605	109668	129758	**108191**	108201	108290
rat99	1211	1232	1505	1483	1419	1301	1220	1217	**1216**
KroA100	21282	21448	21585	44385	25196	22229	21377	21436	**21284**
KroB100	22141	23006	22979	35921	26563	23432	22196	**22173**	22196
KroC100	20749	21583	21826	31290	25343	22108	20923	21103	**20821**
KroD100	21294	21639	21481	34775	24771	23155	21319	21415	**21301**
KroE100	22068	22598	22489	28596	26903	23385	22139	22336	**22137**
rd100	7910	8189	8211	8169	7915	**7910**	**7910**	7946	7914
eil101	629	664	656	668	658	642	647	**630**	640
lin105	14379	14824	14669	53308	18194	16104	14478	14466	**14439**
pr107	44303	45072	45985	208531	53056	46811	45991	46247	**44660**
pr124	59030	62519	60338	183858	66010	**59201**	59751	59475	59562
bier127	118282	122733	121856	210394	142707	189914	121192	120586	**119023**
ch130	6110	6284	6589	6329	7120	6125	6228	6325	**6179**
pr136	96772	102213	103260	103470	105618	97798	101165	100049	**96981**
pr144	58537	59282	59686	225172	71006	59005	59995	60633	**58624**
ch150	6528	6729	6982	6902	7916	**6582**	6608	6665	6585
KroA150	26524	27592	28956	44854	31244	30012	27561	**27315**	27536
KroB150	26130	27572	29450	45374	31407	29192	26867	26981	**26830**
pr152	73682	75834	74562	106180	85616	76710	76327	75980	**74231**
u159	42082	45778	45964	124951	51327	43002	43409	**42511**	43251
rat195	2323	2389	2452	3798	2913	2998	2439	2361	**2330**
d198	15780	15963	16520	78217	17962	23036	17161	16533	**15896**
KroA200	29368	29741	31204	62013	35958	35242	**29735**	29963	29812
KroB200	29437	30516	30265	54570	36412	35636	31103	30199	**29805**
Avg.Gap	0	3.34%	4.95%	22.83%	15.56%	10.06%	2.07%	1.53%	0.76%

Note: results with * are reported from others papers. Bold indicates that the corresponding method is the best among all learning based ones.

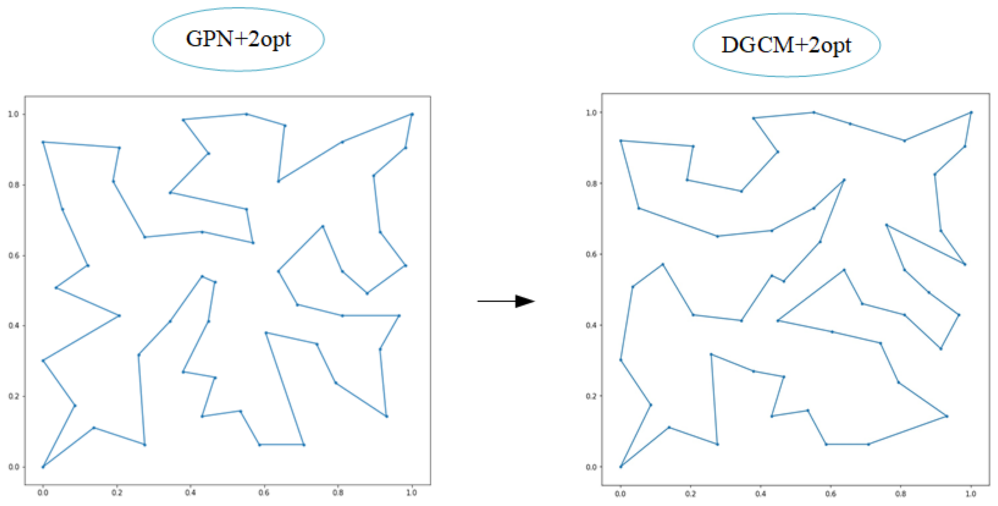

Figure 5. Comparison of GPN and DGCM model visualization on eil51.

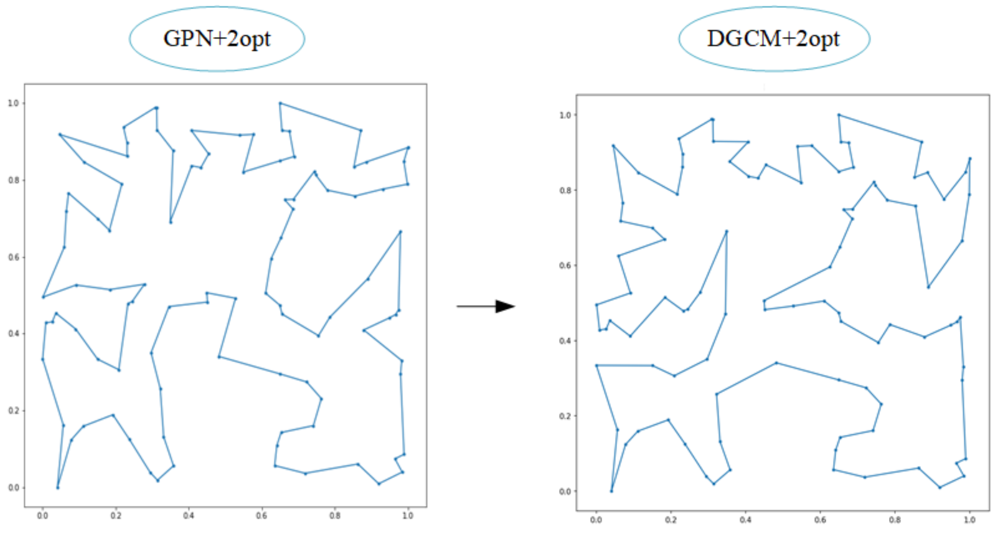

Figure 6. Comparison of GPN and DGCM model visualization on KroA100.

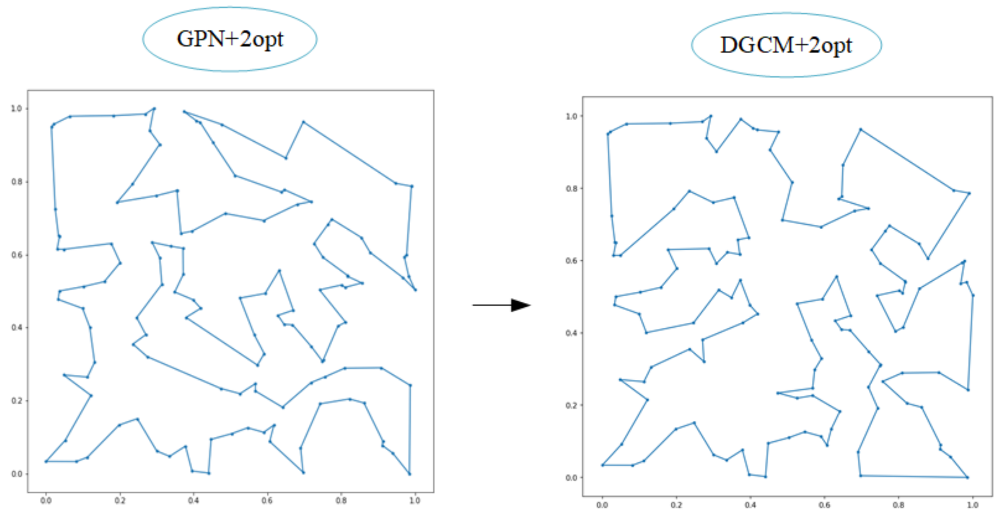

Figure 7. Comparison of GPN and DGCM model visualization on ch130.

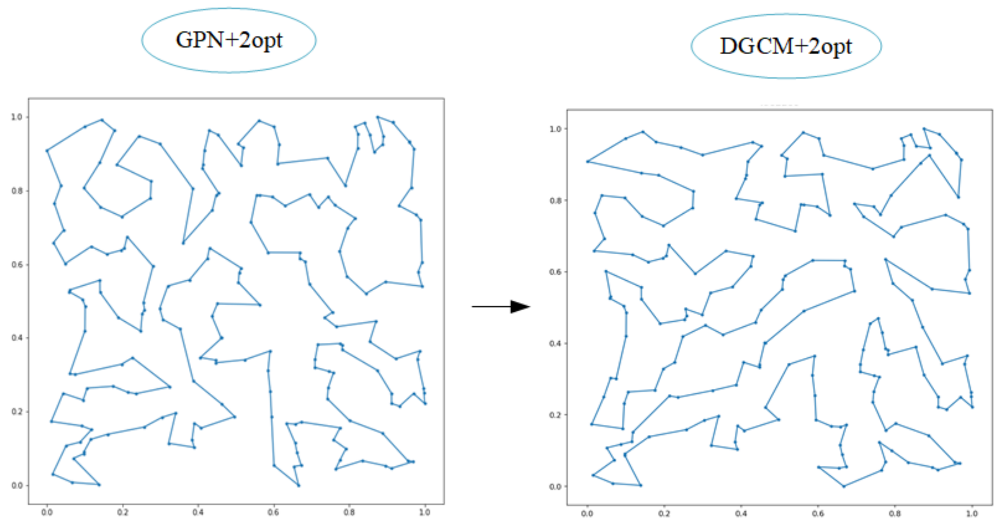

Figure 8. Comparison of GPN and DGCM model visualization on KroB200.

5. Conclusions and future research

In this paper, we have proposed a novel DGCM to learn construction heuristics for large-scale TSP that is trained by DRL. It employs a dynamic encoder-decoder architecture and a Conv-LSTM network to train construction policies at different construction steps. A DPE layer is included in the DGCM; it empowers the decoders with more location information embeddings. The experimental results show that the optimization of our model on TSP surpasses current DRL methods and some traditional algorithms. The solutions of large-scale TSP close to those achieved by professional solving tools with reasonable time. Moreover, the DGCM model generalizes well to TSP of different sizes and even to real-world datasets. In addition, the performance of the DGCM is better than that of the GPN on small-scale and large-scale TSP.

The motivation for using DRL to solve COP may not be to outperform classical methods after sufficient research. Neural networks can be used as a general tool to solve previously unencountered NP-hard problems, especially those for which it is difficult to design heuristic algorithms. In the future, we will adopt DRL-based methods to solve more types of COP. Further, we hope that the DGCM can be extended to solve some of the complex variations of TSP in the real world by hybridization with operational research methods such as TSP with time windows, thereby opening a new era for COP.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (11761042). Moreover, we thank Ma et al. [12] for sharing their source code, which served as the initial basis for our work.

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. M. Bellmore, G. L. Nemhauser, The traveling salesman problem: A survey, *Oper. Res.*, **16** (1968), 538–558. https://doi.org/10.1007/978-3-642-51565-1
2. C. H. Papadimitriou, The euclidean travelling salesman problem is np-complete, *Oper. Res.*, **4** (1977), 237–244. https://doi.org/10.1016/0304-3975(77)90012-3
3. C. William, *World TSP*, 2021. Available from: http://www.sars-expertcom.gov.hk/english/reports/reports.html.
4. R. Bellman, Dynamic programming treatment of the travelling salesman problem, *J. ACM*, **9** (1962), 61–63. https://doi.org/10.1145/321105.321111
5. V. V. Vazirani, *Approximation Algorithms*, Springer Science & Business Media Press, 2013. https://doi.org/10.1007/978-3-662-04565-7
6. Y. Hu, Q. Duan, Solving the TSP by the AALHNN algorithm, *Math. Biosci. Eng.*, **19** (2022), 3427–3488. https://doi.org/10.3934/mbe.2022158
7. J. J. Hopfield, D. W. Tank, "Neural" computation of decisions in optimization problems, *Biol. Cyber.*, **52** (1985), 141–152. https://doi.org/10.1007/BF00339943
8. K. Panwar, K. Deep, Transformation operators based grey wolf optimizer for travelling salesman problem, *J. Comput. Sci.*, **55** (2021), 101454. https://doi.org/10.1016/j.jocs.2021.101454
9. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S. Y. Philip, A comprehensive survey on graph neural networks, *IEEE Trans. Neural Networks Learn. Syst.*, **32** (2020), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386
10. Q. Wang, C. Tang, Deep reinforcement learning for transportation network combinatorial optimization: A survey, *Knowl. Based Syst.*, **233** (2021), 107526. https://doi.org/10.1016/j.knosys.2021.107526

11. Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, *Eur. J. Oper. Res.*, **290** (2021), 405–421. https://doi.org/10.1016/j.ejor.2020.07.063

12. Q. Ma, S. Ge, D. He, D. Thaker, I. Drori, Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning, preprint, arXiv:1911.04936. https://doi.org/10.48550/arXiv.1911.04936

13. O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in *Proceedings of the 29th Concerence on Neural Information Processing System (NIPS)*, **28** (2015), 2692–2700.

14. I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in *Proceedings of the 28th Concerence on Neural Information Processing System (NIPS)*, **27** (2014), 3104–3112.

15. I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, in *Proceedings of the 5th International Conference on Learning Representations*, 2017.

16. H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in *Proceedings of the 31th Concerence on Neural Information Processing System (NIPS)*, **30** (2017), 6351–6361.

17. C. K. Joshi, Q. Cappart, L. M. Rousseau, T. Laurent, X. Bresson, Learning tsp requires rethinking generalization, preprint, arXiv:2006.07054. https://doi.org/10.48550/arXiv.2006.07054

18. W. Kool, H. van Hoof, M. Welling, Attention, learn to solve routing problems, in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

19. Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning improvement heuristics for solving routing problems, in *IEEE Transactions on Neural Networks and Learning Systems*, **(**2021), 1–13. https://doi.org/10.1109/TNNLS.2021.3068828

20. L. Xin, W. Song, Z. Cao, J. Zhang, Multi-decoder attention model with embedding glimpse for solving vehicle routing problems, in *Proceedings of the 35th Conference on Artificial Intelligence (AAAI)*, (2021), 12042–12049.

21. Y. D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, S. Min, Pomo: Policy optimization with multiple optima for reinforcement learning, in *Proceedings of the 34th Concerence on Neural Information Processing System (NIPS)*, **33** (2020), 21188–21198.

22. Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, J. Tang, Learning to iteratively solve routing problems with dual-aspect collaborative transformer, in *Proceedings of the 35th Concerence on Neural Information Processing System (NIPS)*, **34** (2021), 11096–11107.

23. W. Kool, H. van Hoof, J. Gromicho, M. Welling, Deep policy dynamic programming for vehicle routing problems, in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, (2022), 190–213. https://doi.org/10.1007/978-3-031-08011-1_14

24. X. Bresson, T. Laurent, The transformer network for the traveling salesman problem, preprint, arXiv:2103.03012.

25. B. Hudson, Q. Li, M. Malencia, A. Prorok, Graph neural network guided local search for the traveling salesperson problem, preprint, arXiv:2110.05291.

26. L. Xin, W. Song, Z. Cao, J. Zhang, NeuroLKH: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem, in *Proceedings of the 35th Concerence on Neural Information Processing System (NIPS)*, **34** (2021), 7472–7483.

27. W. Chen, Z. Li, C. Liu, Y. Ai, A deep learning model with conv-LSTM networks for subway passenger congestion delay prediction, *J. Adv. Trans.*, **2021** (2021). https://doi.org/10.1155/2021/6645214

28. T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

29. R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Mach. Learn.*, **8** (1992), 229–256. https://doi.org/10.1007/978-1-4615-3618-5_2

30. D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, t al., Certification of an optimal TSP tour through 85,900 cities, *Oper. Res. Lett.*, **37** (2009), 11–15. https://doi.org/10.1016/j.orl.2008.09.006

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

32. Gurobi Optimization, Gurobi optimizer reference manual, 2016. Available from: http://www.gurobi.com.

33. Google, OR-Tools, 2018. Available from: https://developers.google.com.

34. G. Reinelt, Tspliba traveling salesman problem library, *ORSA J. Comput.*, **3** (1991), 376–384. https://doi.org/10.1287/ijoc.3.4.376

AIMS Press