2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) | 978-1-6654-1924-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/WIIAT50758.2020.00102

GGTAN: Graph Gated Talking-Heads Attention Networks for Traveling Salesman Problem

1st Shichao Guo
School of Artificial Intelligence
University of
Chinese Academy of Sciences
Beijing, China
guoshichao18@mails.ucas.ac.cn

2nd Yang Xiao
School of Computer Science and Technology
University of
Chinese Academy of Sciences
Beijing, China
xiaoyang18@mails.ucas.ac.cn

3nd Lingfeng Niu
CAS Research Center on Fictitious
Economy & Data Science, University of
Chinese Academy of Sciences
Beijing, China
niulf@ucas.ac.cn

***Abstract*—Traveling Salesman Problem (TSP) is one of the most typical NP-hard combinatorial optimization problems with a variety of real-life applications. In this paper, we propose a Graph Gated Talking-Heads Attention Networks (GGTAN) trained with reinforcement learning (RL) for tackling TSP. GGTAN can learn characteristic structure information better by introducing talking-heads attention mechanism and a gated convolutional sub-network, which make hidden information moving across between attention heads and control each attention head's importance respectively, unlike recently proposed models which use attention mechanism for solving TSP. Experimental results on TSP up to 100 nodes demonstrate that our model obtains shorter tour lengths than other learning-based methods under the same solve strategy for problem instances of fixed graph sizes, and achieves better generalization on variable graph sizes compared with recent state-of-the-art models on the optimality gap.**

I. Introduction

Traveling Salesman Problem (TSP) is a typical combinatorial optimization problem, which is well-known as an NP-hard problem [1]. The goal of TSP is to find the shortest route that salesman visits each city once and ends in the original city. Designing excellent heuristic algorithms for solving combinatorial optimization problems usually requires specialized knowledge and research time in specific fields. As early as 1954, Dantzig proposed the mathematical planning of the TSP [2]. Usually, the solver is designed by heuristic methods [3], which can search for approximate solutions in a graph containing a large number of nodes, representatives of such advanced solvers include Gurobi [4], Cplex [5], Concorde [6], LKH3 [7] et al.

With the vigorous development and application of deep learning, it brings new inspirations for solving combinatorial optimization problems. In recent years, some learning-based methods proposed for solving combinatorial optimization including TSP, show that deep learning has the potential to learn better heuristics than human-designed methods. In the field of deep learning for solving TSP, Vinyals et al. [8] proposed Pointer Networks based on Recurrent Neural Network (RNN) and trained the model by supervised learning. Bello et al. [9] proposed a reinforcement learning model for Pointer Networks. Dai et al. [10] designed heuristics by Deep Q-Network, and used the structure2vec graph embedding model, a type of Graph Neural Network [11] replaced the RNN structure in Pointer Networks. Most recently, Deudon et al. [12] and Kool et al. [13] borrowed the idea of the attention mechanism proposed by Vaswani [14] in the model of solving TSP, all trained the model by reinforcement learning greedy policy, and Kool's model got close to the optimal TSP solution for up to 100 nodes. Miki et al. [15] and Joshi et al. [16] constructed models based on convolutional neural networks and graph convolutional networks respectively with supervised learning. Qiang et al. [17] proposed Graph Pointer Networks (GPNs) built upon Pointer Networks.

Unlike attention models in [12] [13] recently proposed just based on multi-head attention (MHA) [14] in Graph Attention Networks [18], or others focus on adding different search algorithms into models. In this paper, we focus on learning the characteristic information for the problem with a more powerful attention mechanism and introduce Graph Gated Talking-Heads Attention Networks (GGTAN) trained with reinforcement learning for tackling TSP. GGTAN can learn characteristic structure information better by introducing talking-heads mechanism and a gated convolutional sub-network, which make information moving across attention heads and control each attention head's importance respectively.

We carry out experiments on TSP instances of fixed graph sizes with 20, 50 and 100 nodes. Experimental results show that our model decreases the optimality gap than other learning-based methods under the same solve strategy, and achieves better generalization on variable graph sizes compared with recent state-of-the-art models in terms of solution quality. These results clearly indicate the potential of our new method for solving TSP.

II. Related works

Applying neural networks to solve combinatorial optimization problems can be traced back to 1985, Hopfield & Tank applied Hopfield networks to solve small TSP [19]. The application of deep neural networks to solve TSP started from the seminal work of Pointer Network [8], which based on seq2seq structure, trained in a supervised way. Bello et al. [9] abandoned the idea of supervised learning and proposed a reinforcement learning model for Pointer Network, which based on policy gradients methods [20] and a variant of the A3C algorithm [21].

Dai et al. [10] proposed another way of thinking, they built a GNN [22], a type of structure2vec graph embedding model, which adopted Deep Q-Network to train a node selection heuristic that works within a greedy algorithm framework for solving TSP. Nowak et al. [23] directly applied supervised learning to train a GNN output the solution path as an adjacency matrix, and then used beam search to find the feasible solution. Ma et al. [17] proposed Graph Pointer Networks (GPN), built upon Pointer Networks by introducing a graph embedding layer on the input with capturing relationships between nodes, which trained the model by reinforcement learning and got good performance on large scale issues of TSP.

Miki et al. [15] proposed a model based on convolutional neural networks (CNN), obtained the Good-Edge distribution that may contain the optimal path and calculated the Good-Edge value for neighborhood search. Joshi et al. [16] used graph convolutional neural networks (GCN) to represent the graph structure of TSP trained in a supervised way, which shortens the gap with the special solver for operations research, with adding a highly parallel beam search to obtain solutions, compared to some autoregressive models [8]–[10], [12] in the past.

Moreover, Kaempfer & Wolf [24], Deudon et al. [12] and Kool et al. [13] all adopted the idea of the attention mechanism proposed by Vaswani [14]. The model trained by Kaempfer & Wolf based on Permutation Invariant Pooling Networks focusing on multiple traveling salesman problems (mTSP). The model performance of Deudon et al. [12] relied on an additional 2-opt based local search. Kool et al.[17] replaced the RNN based sequential structures in Seq2Seq with the attention modules in the Transformer architecture [18], and achieved good performances on TSP with 20, 50 and 100 nodes size with a reinforcement learning with greedy rollout baseline.

III. Problem Formulation and Preliminaries

In this work, we focus on solving 2D Euclidean TSP. It can be described as: there are multiple cities on the map and distance between any two cities is known. The traveling salesman needs to start from a certain city and pass through all cities, each city only passes once, and finally returns to the departure city. The goal is to calculate the shortest route traveled by the traveling salesman.

TSP can be regarded as a sequential decision problem. The encoder-decoder architecture is an effective framework to solve this kind of problem. In deep learning, the encoder-decoder architecture is a common framework with many applications designed by it, which can be used structuring a flexible model to meet the needs of the problem. As shown in Fig. 1, the encoder-decoder architecture uses the encoder to extract the structural characteristics of the input instance, the decoder gradually constructs the solution sequence. In a one-step construction process, the decoder predicts the distribution of the nodes, then selects a node and appends it to the end of the partial solution path.

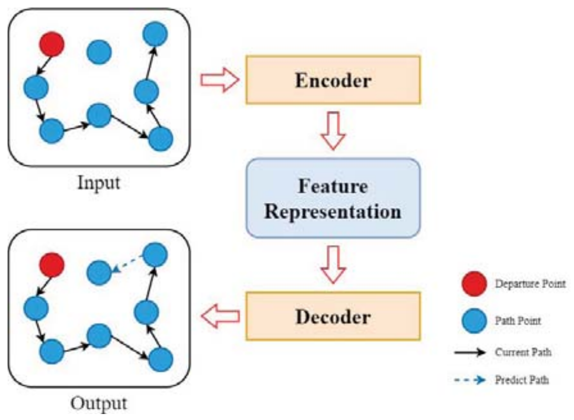

Fig. 1. The encoder-decoder architecture for TSP.

For an input instance X of TSP, X can be thought as a full connection graph consisting of n nodes, node $i \in \{1, ..., n\}$ has characterized by x_i. Define the optimal solution path as $\pi = (\pi_1, ..., \pi_n)\,,\ \pi_t \in \{1, ...n\}$. Therefore, for parameter θ and input instance X, the corresponding solution probability $p_\theta\left(\pi|X\right)$ can be defined as:

$$p_\theta\left(\pi|X\right) = \prod_{t=1}^{T} p_\theta\left(\pi_t|X, \pi_{1:t-1}\right) \tag{1}$$

IV. Graph Gated Talking-Heads Attention Networks

We propose a Graph Gated Talking-Heads Attention Network (GGTAN) for approximately solving TSP, in which architecture consists of an encoder and decoder component.

A. *Encoder*

In a typical Transformer architecture, node features are embedded in context of graph in the encoder. For TSP, node feature x_i has input dimension $d_x = 2$, in hidden layer, for each node embedding $h_i^{(0)}$ has dimension $d_h = 128$, which is computed through a learnable linear transformation. The parameters of the linear transformation are $W \in \mathbb{R}^{d_n \times d_x}$ and $b \in \mathbb{R}^{d_h}$, $h_i^{(0)}$ is calculated as follows:

$$h_i^{(0)} = Wx_i + b, \tag{2}$$

The initial node embeddings for each TSP instance are sent to the graph attention network and updated $N = 3$ times with N graph attention layers. The encoder ultimately gets the node embedding $h_i^{(N)}$, as well as the graph embedding $\bar{h}^{(N)} = 1/n\sum_{i=1}^{n} h_i^{(N)}$, which will all be input to the decoder.

In the encoder, each attention layer is composed of three sublayers, one is the talking-heads attention sublayer, one is the gated convolutional sublayer, and the last one is the fully connected feed-forward sublayer.

Talking-Heads Attention Multi-head attention proposed in [14] is widely used and achieves good results in many models of deep learning, in which the features learned by different attention heads independently in separate computations, and sum up at the end. Inspired by the research of Shazeer et al. [25], we adopt talking-heads attention (THA) mechanism, which breaks separation in traditional multi-head attention that attention scores calculated by different attention heads are isolated from each other. In THA, we insert two additional learned linear projections P_c and P_w, which transform attention compatibilities and the attention weights respectively, so that feature information shared between separated attention heads.

In attention layer $l \in \{1, ..., N\}$, $h_i^{(l)}$ is node embedding of node i, the output $\left\{h_0^{(l-1)}, ..., h_n^{(l-1)}\right\}$ of layer $l-1$ is the input of layer l . The weight of message value that x_i received from its neighbor nodes depends on its query value and neighbors' key value. For x_i, query value (q), key value (k) and message value (v) are calculated by projecting the embedding h_i:

$$q_{im}^{(l)} = W_m^Q h_{im}^{(l-1)},\ k_{im}^{(l)} = W_m^K h_{im}^{(l-1)},\ v_{im}^{(l)} = W_m^V h_{im}^{(l-1)}, \tag{3}$$

Here, the total number of attention heads is $M = 8$, the dimensions of k and v represented by d_k , d_v where $d_k = d_v = d_h/M$. For each attention head $m \in \{1, ..., M\}$ obtains $q_{im}^{(l)} \in \mathbb{R}^{d_k}$, $k_{im}^{(l)} \in \mathbb{R}^{d_k}$, $v_{im}^{(l)} \in \mathbb{R}^{d_v}$ thorough the learnable parameter $W_m^Q \in \mathbb{R}^{d_k \times d_h}$, $W_m^K \in \mathbb{R}^{d_k \times d_h}$, $W_m^V \in \mathbb{R}^{d_v \times d_h}$. The attention compatibility $u_{ijm}^{(l)}$ can be calculated as dot-product attention:

$$u_{ijm}^{(l)} = \begin{cases} \frac{\left(q_{im}^{(l)}\right)^T k_{jm}^{(l)}}{\sqrt{d_k}} & \text{if } i \text{ adjacent to } j, \\ -\infty & \text{otherwise.} \end{cases} \tag{4}$$

After this step, all attention heads talk with each other by using a learned linear projections P_c and P_w with parameter $W^{p_c} \in \mathbb{R}^{M \times T}$ and $W^{p_w} \in \mathbb{R}^{T \times M}$ to calculate weights $w_{ijm}^{(l)}$ using a softmax in the computational process. Here T refers to the number of attention head for THA ($T > M, t \in \{1, ..., T\}$), $\|$ refers to sequential concatenation operation, and THA attention vectors can be calculated as:

$$h_{im}^{\prime(l)} = \sum_{j=1}^{n} w_{ijm}^{(l)} v_{jm}^{(l)}, \tag{5}$$

$$\text{THA}_i^{(l)} \left(h_0^{(l-1)}, \ldots, h_n^{(l-1)}\right) = \mathop{\Big\|}_{m=1}^{M} h_{im}^{\prime(l)}. \tag{6}$$

Gated Convolutional Aggregator Usually, when aggregating the multiple attention heads in the end, only simple splicing operations or average operations are used. But not all of these subspaces between the central node and its neighbors explored by multiple attention heads are equally important. In order to reduce or even eliminate the influence of these kinds of useless representations, inspired by the research in [26], we add a soft gated convolutional aggregator in talking-heads attention network. The gated values range between 0 to 1, which indicate the importance of each attention head. The gated convolutional aggregator can be represented as:

$$\begin{aligned} \text{GatedTHA }_i \left(h_0, \ldots, h_n\right) &= G_i \text{THA}_i \left(h_0, \ldots, h_n\right) \\ &= \mathop{\Big\|}_{m=1}^{M} g_i^{(m)} h_i^{(m)}, \end{aligned} \tag{7}$$

Here, $g_i^{(m)} \in (0, 1)$ represents a soft gate value of the mth attention head for x_i.

We use a convolutional network, which is composed of average pooling and maximum pooling to generate the gate value. We send the current node embedding and neighbor node features as input, by using a fully-connected network during the execution of maximum pooling operations, and a fully-connected network contains sigmoid activation function in the process of final aggregation. Before the maximum pooling, the neighbor node features are converted to $d_m = 64$ dimension vector through the learnable parameter θ_m. Finally, the concatenated features aggregate and generate M gated values through the learnable parameter θ_g.

Feed-Forward Sublayer In this sublayer, talking-heads attention vector is processed through a full connect feed-forward (FF) network, skip-connection, and ReLU activation function, for x_i:

$$\hat{h}_i^{(\ell)} = \tanh\left(h_i^{(\ell-1)} + \text{GatedTHA}_i^{(l)} \left(h_0^{(\ell-1)}, ..., h_n^{(\ell-1)}\right)\right), \tag{8}$$

$$\text{FF}\left(\hat{h}_i^{(\ell)}\right) = W_1^F \text{ReLu}\left(W_0^F \hat{h}_i^{(\ell)} + b_0^F\right) + b_1^F, \tag{9}$$

$$h_i^{(\ell)} = \tanh\left(\hat{h}_i^{(\ell)} + \text{FF}\left(\hat{h}_i^{(\ell)}\right)\right). \tag{10}$$

Here, $W_0^F \in \mathbb{R}^{d_F \times d_h}$, $W_1^F \in \mathbb{R}^{d_F \times d_h}$, $b_0^F \in \mathbb{R}^{d_F}$, $b_1^F \in \mathbb{R}^{d_F}$, where $d_F = 4 \times d_h$.

B. Decoder

In the decoder, the decoding step of TSP is sequential. For every construction step $t \in \{1, ..., n\}$, current node selects the next node to go based on the partial solution and the current node embedding of each node. The new context vector is constructed as follows:

$$h_c' = \begin{cases} \left[\bar{h}_t; h_0^N\right] & \text{if } t = 1, \\ \left[\bar{h}_t; h_{\pi_{t-1}}^N\right] & \text{if } t > 1, \end{cases} \tag{11}$$

Here, $[;]$ refers to concatenation operation, $h_{\pi_{t-1}}^N$ refers to node embedding at step $T = t - 1$, $\bar{h}_t$ refers to graph embedding, which is the average vector of the node embedding belong to nodes that are not accessed at $T = t$. The decoder context vector h_c is calculated by MHA, which is similar to the encoder, but only query value per head is calculated, and the parameters are not shared with the encoder.

probability calculation To calculate the output probability $p_\theta\left(\pi_t | X, \pi_{1:t-1}\right)$, the model uses a single-head attention layer, only the dot-product attention compatibility u_j is calculated:

$$q = W^Q h_c,\ k_j = W^K h_j^N, \tag{12}$$

TABLE I
PERFORMANCE OF OUR MODEL COMPARED TO NON-LEARNED BASELINES AND STATE-OF-THE-ART METHODS FOR VARIOUS TSP INSTANCE SIZES

Method	Type	TSP20			TSP50			TSP100		
		Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time
Concorde [Applegate et al., 2006]	Solver	3.83	0.00%	1m	5.69	0.00%	2m	7.76	0.00%	3m
Gurobi [Gurobi Optimization, 2015]	Solver	3.83	0.00%	7s	5.69	0.00%	2m	7.76	0.00%	17m
LKH3 [Helsgaun, 2017]	H	3.83	0.00%	18s	5.69	0.00%	5m	7.76	0.00%	21m
OR Tools [Google, 2016]	H,S	3.85	0.37%	-	5.80	1.83%	-	7.99	2.90%	-
2-OPT [Aarts et al., 2003]	H	3.96	3.24%	-	6.12	7.39%	-	8.51	9.71%	-
Chr.f. + 2-OPT [Vinyals et al., 2016]	H, 2-OPT	3.85	0.37%	-	5.79	1.65%	-	8.48	9.39%	-
Nearest Insertion	H,G	4.33	12.91%	1s	6.78	19.03%	2s	9.46	21.82%	6s
Random Insertion	H,G	4.00	4.36%	0s	6.13	7.65%	1s	8.52	9.69%	3s
Farthest Insertion	H,G	3.93	2.36%	1s	6.01	5.53%	2s	8.35	7.59%	7s
Nearest Neighbor	H,G	4.50	17.23%	1s	7.00	22.94%	2s	9.68	24.73%	2s
PtrNet [Vinyals et al., 2015]	SL,G	3.88	1.15%	-	7.66	34.48%	-	-	-	-
PtrNet [Bello et al., 2017]	RL,G	3.89	1.42%	-	5.95	4.46%	-	8.30	6.90%	-
S2V-DQN [Dai et al., 2017]	RL,G	3.89	1.42%	-	5.99	5.16%	-	8.31	7.03%	-
EAN [Deudon et al., 2018]	RL,G	3.86	0.66%	2m	5.92	3.98%	5m	8.42	8.41%	8m
AM [Kool et al., 2019]	RL,G	3.85	0.34%	1s	5.80	1.76%	2s	8.12	4.53%	6s
GCN [Joshi et al., 2019]	SL,G	3.86	0.60%	6s	5.87	3.10%	55s	8.41	8.38%	6m
GPN [Ma et al., 2019]	RL,G	3.87	0.96%	1s	5.96	4.56%	4s	8.51	9.70%	14s
GGTAN [Ours]	**RL,G**	**3.84**	**0.21%**	**1s**	**5.77**	**1.59%**	**3s**	**8.02**	**3.25%**	**7s**
GGTAN [Ours]	**RL,BS**	**3.83**	**0.00%**	**3.7m**	**5.72**	**0.52%**	**24m**	**7.92**	**1.91%**	**1.5h**

$$u_j = \begin{cases} C \cdot \tanh\left(\frac{q^T k_j}{\sqrt{d_k}}\right) & \text{if } j \notin \pi_{1:t-1}, \\ -\infty & \text{otherwise }, \end{cases} \quad (13)$$

Here, C is used to clip the result with the value between$[-C, C]$ $(C = 10)$. Then, the final output probability vector P is calculated by softmax:

$$p_j = p_\theta\left(\pi_t = x_j | X, \pi_{1:t-1}\right) = \frac{e^{u_j}}{\sum_{j'=1}^{n} e^{u_j}}. \quad (14)$$

V. MODEL TRAINING

As TSP solving process can be viewed as Markov Decision Processes (MDP), we use REINFORCE algorithm [20] training our model in this paper. For an instance X of TSP, the goal of training is the length of the solution path π, the gradient of the parameter θ can be defined as:

$$\nabla_\theta J(\theta|X) = \mathbb{E}_{\pi\sim p_\theta(.|X)}\left[\left(L(\pi|X) - b(X)\right)\nabla_\theta \log p_\theta(\pi|X)\right], \quad (15)$$

Where $L(\pi|X)$ refers to the length of the TSP solution path π, $b(X)$ is the solution baseline constructed by greedy rollout same as [7], represents the tour length of the greedy solution. $b(X)$ estimates the expected solution path length $\mathbb{E}_{\pi\sim p_\theta(.|X)} L(\pi|X)$, and can effectively reduce the variance of the gradient and accelerate the convergence rate.

The model training process uses Adam optimizer [28]. Through Monte Carlo sampling, the gradient of the parameter θ can be approximately expressed as:

$$\nabla_\theta J(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\left[\left(L(\pi_i^s|X_i) - L(\pi_i^g|X_i)\right)\nabla_\theta \log p_\theta(\pi_i^s|X_i)\right], \quad (16)$$

Here B stands for batch size, π_i^s and π_i^g represent the solution path derived from sampling rollout and greedy rollout.

VI. EXPERIMENTS

A. Datasets and parameters

The datasets are generated by Concorde [6] on fly, which is divided into three subsets with different scales of $n = 20, 50, 100$ respectively, each containing 10,000 pairs of TSP instances and solution path sequences. In our model, GGTAN encoder has 3-attention layer with 128-dimensional embeddings and hidden states, 64-dimensional gate states, 8 attention heads, 32 talking-heads per layer. Model is trained using the Adam optimizer [27] with a fixed learning rate of 10^{-4} for 100 epochs with mini-batches of size 512, where the epoch size is 1,000,000. All experiments are conducted by a single GPU (Nvidia 1080Ti). We use *Optimality gap* same as in [13] evaluating the performance of models.

B. Results and Discussions

Comparison Results In Table I, we compare the performance of our model with other models on TSP sizes of 20, 50 and 100 respectively. Each row in the table represents an algorithm model, which is divided into four categories according to the training method, namely Solver, Heuristic algorithm (H), Supervised Learning (SL) and Reinforcement Learning (RL). The whole table is divided into two regions. Firstly, Solver and Heuristic non-learned algorithms. Here, Concorde [6], Gurobi [4] are special solvers for combination optimization, LKH3 [7] is the best heuristic algorithm solver currently available, OR Tools [28] is a solver with excellent performance produced by Google. Meanwhile, it also includes 2-OPT [29] and other heuristic non-learning algorithms [8], [13]. Secondly, learning-based models using greedy policy (G). Some parts of result is directly taken from [16] (the order of the results is slightly changed), while unavailable items are marked as ”-”.

As shown in Table I, combination optimization solvers including Concorde [6], Gurobi [4] and heuristic solver LKH3 [7] get optimal solution, while other non-learned algorithms

TABLE II
THE GENERALIZATION PERFORMANCE OF OUR MODEL ON VARIABLE TSP INSTANCE SIZES

Method	Type	TSP20			TSP50			TSP100		
		Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time
Concorde [Applegate et al., 2006]	Solver	3.83	0.00%	1m	5.69	0.00%	2m	7.76	0.00%	3m
AM-TSP20 [Kool et al., 2019]	RL,G	3.85	0.34%	1s	5.95	4.53%	1s	9.01	16.11%	6s
AM-TSP50 [Kool et al., 2019]	RL,G	3.89	1.43%	1s	5.80	1.76%	1s	8.18	5.3%	6s
AM-TSP100 [Kool et al., 2019]	RL,G	4.36	13.88%	1s	5.97	4.91%	1s	8.15	4.53%	6s
GPN-TSP20 [Ma et al., 2019]	RL,G	3.87	0.96%	1s	8.95	57.29%	4s	14.88	91.75%	15s
GPN-TSP50 [Ma et al., 2019]	RL,G	4.16	8.63%	1s	5.96	4.56%	4s	10.01	28.99%	15s
GPN-TSP100 [Ma et al., 2019]	RL,G	4.31	12.64%	1s	6.26	9.98%	4s	8.51	9.70%	15s
GGTAN-TSP20 [Ours]	**RL,G**	**3.84**	**0.21%**	**1s**	**5.96**	**4.66%**	**2s**	**8.94**	**15.15%**	**7s**
GGTAN-TSP50 [Ours]	**RL,G**	**3.88**	**1.25%**	**1s**	**5.77**	**1.59%**	**2s**	**8.09**	**4.12%**	**7s**
GGTAN-TSP100 [Ours]	**RL,G**	**4.13**	**7.83%**	**1s**	**5.89**	**3.41%**	**2s**	**8.02**	**3.25%**	**7s**

[8], [13], [29] and OR tools (except on TSP100) [28] perform poorly than GGTAN. Compared with learning-based models using greedy rollout policy similarly, GGTAN achieves the improvement of accuracy. In particular, compared with the GPN [17] and Greedy GCN [16], GGTAN performances superior on result quality and solving time of all fixed scale of problems. Due to adopting powerful but more complex attention mechanism, GGTAN gets lower gap with the cost of solving time increases compared Attention Model (AM) [13], but still performances better than other greedy policy models.

What's more, we can see the hybrid model of GGTAN with adding a beam search algorithm has higher overall solution accuracy but longer solving time compared with GGTAN based on greedy policy. So the model fusion search algorithm can be regarded as a strategy that sacrifices time cost for getting accuracy.

Generalization to variable problem sizes The generalization ability to solve TSP on variable sizes is a very important property of the learning-based model, which allows us to extend solving large-scale TSP on the basis of effective model training for a small TSP instance. Specifically, the models trained with instances with 20, 50 and 100 nodes are denoted as GGTAN-TSP20, GGTAN-TSP50 and GGTAN-TSP100 respectively. GGTAN-TSP20 is tested on instances with 50 and 100 nodes, GGTAN-TSP50 is tested on instances with 20 and 100 nodes, and GGTAN-TSP100 is tested on instances with 20 and 50 customer nodes, respectively.

Table II shows the comparison of the generalization performance of GGTAN to state-of-the-art models proposed by Kool et al. [13] and Ma et al. [17] which have outstanding performance on solving fixed problem sizes in recent years. The results show that GGTAN trained with small instances (n = 20) has better performance than baseline methods on large instances (n = 50, n = 100), as well as trained with large instances (n = 100) on solving small sizes of problem. The reason why our model has a good generalization performance may be as follows. Theoretically, when enough Gaussian distributions are superimposed, Gaussian mixture distribution can approximate any probability distribution. Using talking-heads attention mechanism in GGTAN enables the hidden feature information to interact between the attention heads, form a Gaussian mixture distribution enhances the expression ability. Meanwhile, the gated convolutional aggregator can effectively control the importance of each attention head and filter out some less-important information.

VII. CONCLUSION

In this work, we propose a Graph Gated Talking-Heads Attention Network (GGTAN) framework for approximately solving 2D Euclidean TSP. Experimental results on TSP demonstrate that GGTAN obtains a lower optimality gap of solutions on fixed problem sizes and a better generalization ability on variable problem sizes compared with recent state-of-the-art models. The key to improvement is that the structural features of instances learned through a strengthened attention mechanism by introducing talking-heads mechanism and a gated convolutional sub-network, more hidden and useful structure information is taken into account, for constructing a better solution for TSP. In the future, we shall explore incorporating transfer learning and efficient heuristic algorithm into our framework in order to generalize to larger-scale TSP problems and some real-world TSP or TSP variants.

ACKNOWLEDGEMENT

This work was partially supported by the National Natural Science Foundation of China (Grant No. 11991021, 11671379), and Beijing Academy of Artificial Intelligence (BAAI).

REFERENCES

[1] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," Theoretical Computer Science, vol. 4, no. 3, pp. 237-244, 1977.
[2] G. Dantzig, R. Fulkerson, and S. Johnson,"Solution of a large-scale traveling-salesman problem," Journal of the operations research society of America, vol. 2, no. 4, pp. 393-410, 1954.
[3] M. Junger, G. Reinelt, and G. Rinaldi, Handbooks in OR&MS: Chapter 4 the Traveling Salesman Problem. Holland: Elsevier Sci, pp. 225-330, 1995.
[4] G. O. Inc., "Gurobi optimizer reference manual," 2014.
[5] I. I. Cplex, "V12.1: User's manual for CPLEX," IBM Corp., vol. 46, no. 53, pp. 157, 2009.
[6] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, The traveling salesman problem: a computational study. Princeton university press, 2006.
[7] K. Helsgaun, An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Roskilde: Roskilde University, 2017.
[8] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in Proc. Advances in Neural Inf. Process. Syst. (NeurIPS), 2015, pp. 2692-2700.

[9] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” in Proc. Int. Conf. Learn. Representations (ICLR), 2017.
[10] H. Dai, E. Khalil, Y. Zhang, B. Dilkina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” in Proc. Advances in Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 6348–6358.
[11] H. Dai, B. Dai, and L. Song, “Discriminative embeddings of latent variable models for structured data,” in Proc. 33th Int. Conf. on Mach. Learn. (ICML), vol. 48, 2016, pp. 2702–2711.
[12] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, “Learning heuristics for the tsp by policy gradient,” in Proc. Int. Conf. Integration Constraint Program., Artif. Intell., Operations Res. Springer, 2018, pp. 170–181.
[13] W. Kool, H. Van Hoof, and M. Welling, “Attention, learn to solve routing problems!” in Proc. Int. Conf. Learn. Representations (ICLR), 2019.
[14] A. Vaswani, A. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Advances in Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 5998-6008.
[15] S. Miki, D. Yamamoto, and H. Ebara, “Applying deep learning and reinforcement learning to traveling salesman problem,” in Proc. IEEE Int. Conf. Computing, Electronics & Communications Engineering (iCCECE), aug 2018, pp. 65-70.
[16] C. K. Joshi, T. Laurent, and X. Bresson, “An efficient graph convolutional network technique for the travelling salesman problem,” 2019. [Online]. Available: arXiv:1906.01227
[17] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, “Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning,” 2019. [Online]. Available: arXiv:1911.04936.
[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in Proc. Int. Conf. Learn. Representations (ICLR), 2018.
[19] J. J. Hopfield, and D. W. Tank,“‘Neural’ computation of decisions in optimization problems,” Biological cybernetics, vol. 52, no. 3, pp.141-152, 1085.
[20] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. learn., vol. 8, no. 3-4, pp. 229–256, 1992.
[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in Proc. Int. Conf. Mach. Learn. (ICML), 2016, pp. 1928–1937.
[22] H. Dai, B. Dai, and L. Song, “Discriminative embeddings of latent variable models for structured data,” in Proc. Int. Conf. Mach. Learn. (ICML), 2016, pp. 2702–2711.
[23] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, “A note on learning algorithms for quadratic assignment with graph neural networks,” in ICML Workshop on Principled Approaches to Deep Learning (PADL), 2017.
[24] Y. Kaempfer and L. Wolf, “Learning the multiple traveling salesmen problem with permutation invariant pooling networks,” 2018, [Online]. Available: arXiv:1803.09621.
[25] N. Shazeer, Z. Lan, Y. Cheng, N. Ding, and L. Hou, “Talking-Heads Attention. [Online]. Available: arXiv:2003.02436.
[26] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D. Y. Yeung, “GaAN: Gated attention networks for learning on large and spatiotemporal graphs,” in Proc. 34th Conf. Uncertainty in Artificial Intelligence (UIA), 2018, pp. 339-349.
[27] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learn. Representations (ICLR), 2015
[28] O.-t. Google Inc., “OR-tools: Google optimization tools,” 2015.
[29] E. Aarts, E. H. Aarts, and J. K. Lenstra, Local search in combinatorial optimization. Princeton University Press, 2003.