Local Topological Information as A Powerful Enhancer for Generalizable Neural Method in Travelling Salesman Problem

Xiaoxin Bai
School of Computer Science and Engineering, Southeast University, Suzhou, China
baixiaox0@gmail.com

JunYang Yang
School of Cyber Science and Engineering, Southeast University, Suzhou, China
230239714@seu.edu.cn

Shengchao Yuan
School of Computer Science and Engineering, Southeast University, Suzhou, China
220204707@aa.seu.edu.cn

Yinghao Zhang
School of Electronics and Information Engineering, South China University of Technology, Guangzhou, China
zhangyh.1998cn@gmail.com

Hanqian Wu†
School of Cyber Science and Engineering, Southeast University, Suzhou, China
hanqian@seu.edu.cn

ABSTRACT

The generalization challenges faced by neural methods in solving the Travelling Salesman Problem (TSP) have attracted substantial attention. Current neural approaches predominantly rely on Deep Neural Networks with global receptive fields to capture global features, often overlooking the critical role of local topological information. To address this limitation, we propose a Global & Local Encoder (G&L-Encoder) that efficiently integrates local and global information within the feature space. The G-Encoder, with global receptive field, takes charge of capturing global features and facilitating the fusion of global and local information, while the L-Encoder, with its local neighborhood receptive field, extracts finer-grained features specific to each node's neighborhood, serving as an essential complement to the global one. Furthermore, we propose a simple yet efficient supervised learning framework, leveraging the recursive optimality inherent in optimal solutions to dig into their implicit knowledge. Extensive experiments on the TSP demonstrate that our method significantly improves generalization performance and achieves state-of-the-art results on medium-scale TSP tasks. Our source code is available at https://github.com/bxx-seu/GL-POMO.

KEYWORDS

Neural Combinatorial Optimization; Construction Heuristics; Routing Problem; Travelling Salesman Problem; Supervised Learning

ACM Reference Format:
Xiaoxin Bai, JunYang Yang, Shengchao Yuan, Yinghao Zhang, and Hanqian Wu†. 2025. Local Topological Information as A Powerful Enhancer for Generalizable Neural Method in Travelling Salesman Problem. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

†Corresponding authors.

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

The Travelling Salesman Problem (TSP) is a well-known NP-hard problem with extensive applications across various domains. For decades, researchers have devoted considerable effort to developing more efficient solvers for TSP, with Concorde being one of the most notable [8], employing exact algorithms based on cutting-plane methods. Due to the NP-hardness of TSP, the exact algorithms suffer from scalability issues [6]. Consequently, heuristic approaches, such as the Lin-Kernighan heuristic (LKH) [13], have been adopted to find near-optimal solutions within limited time. These heuristics aim to strike a balance between solution quality and computational cost to meet the growing demand for real-time conditions. Typically, they still rely on substantial domain-specific expertise.

Recently, the neural methods based on Machine Learning (ML) have been adapted to solve routing problems, including TSP, which can learn or discover heuristics automatically instead of expertise [1, 7, 24, 26, 27, 34, 37, 43]. Compared to traditional algorithms, neural method can greatly reduce computational costs and development expenses while delivering desirable solutions, making this a promising avenue for further research.

While neural methods demonstrate promising results, existing neural methods generally suffer from the generalization issue [28]. Numerous studies have attempted to address this problem by introducing various techniques, such as meta-learning [33, 48], knowledge distillation [2], and hierarchical architecture [46]. These frameworks have significantly improved the generalization capabilities of models, both in terms of cross-distribution or cross-size. However, an intriguing phenomenon has emerged: models, specifically trained to enhance generalization often perform distinctly worse on smaller and simpler instances compared to their baseline models. This observation prompts a critical reassessment of *whether current mainstream model architectures contain sufficient information to support a generalizable neural method.*

Existing neural methods heavily rely on the representation capabilities of Deep Neural Networks with global receptive fields, e.g. Transformer [24, 38], Graph Neural Network [4, 21]. Many studies have demonstrated that global features are capable of helping the policy escape local optima [21, 22, 24, 25]. However, the lack of local topological information forces the model to memorize all details

of the instance via global receptive field. This, at least in part, may explain why the neural methods for routing problems struggle with generalization. They tend to overfit to the training distributions, and even slight perturbations to the global characteristics of instances can degrade performance. Notably, the policy relying solely on local information without the global receptive fields may easily degenerate into a greedy strategy. Therefore, a more reasonable approach is to treat local information as the complement to global one.

On the other hand, the sparse reward issue in REINFORCE learning [41], increases the computational cost of model training. For certain routing problem, like TSP, it is affordable to pre-compute an adequate number of high-quality solutions for small-scale instances using traditional algorithms. These solutions, meticulously picked from the exponential discrete solution space, encompass abundant information about instance distribution representations and heuristic strategies. By effectively leveraging these pre-computed solutions, we can mitigate the sparse reward issue and enable more efficient model training.

In this paper, we introduce the local topological information as a crucial complement to global features by designing a Global & Local Encoder architecture, termed G&L-Encoder, which efficiently integrates local and global information within the feature space. The G-Encoder, with global receptive field, takes charge of capturing global features, similar to most existing methods [22, 24, 25, 43], while facilitating the fusion of global and local information. The L-Encoder, with its local neighborhood receptive field identified by K-Nearest Neighbor (KNN), extracts fine-grained features specific to each node's neighborhood, serving as an essential complement to the global one. In order to leverage the limited high-quality solutions, we design Sampled Steps Supervised Learning (SSSL), which estimates the gradients by sampling several decision steps (each step representing the selection of the next node based on the current partial tour) from the trajectories of high-quality solutions. The SSSL avoids the need to construct the entire tour during training, reducing computational resource expenditure. The underlying principle is that each decision contributing to the optimal solution is itself optimal. Therefore, learning from these individual decisions is effective for model training and also significantly expands the training labels. Besides, symmetry-based augmentation methods, inspired by both the solution and problem symmetries [22], are employed to further improve the utilization of limited high-quality solutions.

Our contributions are summarized as follows:

- We reveal the significance of local topological information in designing a generalizable neural method for TSP. To accomplish this, we propose a G&L-Encoder architecture that adeptly integrates global and local information by embedding our local policies within a global Transformer-based encoder.
- We design a simple yet efficient training framework, Sampled Steps Supervised Learning (SSSL) that enables the model to learn from individual decisions within high-quality solutions.
- Extensive experiments demonstrate that our method significantly improves the generalization against the backbone models as well as other existing generalization methods on TSP across various distributions and sizes, achieving state-of-the-art results on medium-scale TSP tasks.

2 RELATED WORKS

Neural Methods There have been a soaring number of studies trying to solve routing problems using ML algorithms. According to the way of constructing the solution, most of the neural methods could be divided into two categories: 1) *Construction Heuristics:* The solution is generated by sequentially appending an accessible node to the current partial tour until satisfying all constraints and demands. Vinyals et al. [39] proposed a Pointer Network to solve TSP with supervised learning. Bello et al. [1] then brought REINFORCE algorithm [41] to this field. Kool et al. [24] proposed the attention model (AM) adapted from the Transformer architecture [38] to solve a wide range of Combinatorial Optimization Problems (COPs), which achieved a great improvement on various tasks. The AM has become the *de-facto* standard method. Kwon et al. [25] proposed the policy optimization with multiple optima (POMO) relying on problem-specific solution symmetries for COPs to further improve upon AM. Extended from POMO, Kim et al. [22] designed a general-purpose symmetric learning scheme (Sym-NCO), which could leverage universal symmetries in various COPs and solutions. Besides, Graph neural networks are also utilized to solve TSP [17, 21, 31, 34]. There have been other works proposed to improve upon the above [19, 45]. 2) *Improvement Heuristics:* An initial complete solution is iteratively refined by the learned heuristics until the termination condition is met. In this line of research, the learned heuristics are usually used to control the classical local search methods [5, 7, 29, 32, 42] or to provide critical information for traditional meta-search methods [15, 23]. Generally, improvement methods are able to achieve a better solution than construction ones, but at the expense of much longer inference time. In this paper, we focus on the construction method.

Generalization issue The generalization issue has attracted more and more attention, which severely hinders the practical application of the neural methods [20, 28]. Recently, many preliminary attempts have been made to tackle this issue. Among the early attempts, Xin et al.[44] exploited the generative adversarial network to adaptively generate hard-to-solve instances for training the model. Wang et al.[40] introduced the game theory to learn the trainable solver and instance generators simultaneously. Zhang et al.[47] also tried to design an adaptive instance generator, which could dynamically adjust the hardness of instances according to the current model. Other than data perspective, many works are devoted to introducing or devising suitable training frameworks to assist the trainable model in learning generic strategies. Jiang et al.[18] and Bi et al.[2] introduced group DRO (Distributionally Robust Optimization) [36] and knowledge distillation [14] separately to enhance model robustness. Manchanda et al.[33] and Zhou et al.[48] exploited meta-learning techniques to learn the model on a variety of tasks with various distributions and sizes. Different from above, Luo et al.[30] designed a Light Encoder and Heavy Decoder (LEHD) architecture for cross-size generalization, which could adapt to the situation of changeable size of available nodes in the decoding phase. Gao et al. [11] proposed an ensemble model

comprised of global policy and local policy, which integrates the two policies by adding up the action scores computed by global and local policy respectively. Ye et al. [46] designed a unified hierarchical framework that could decompose large routing problems into several Shortest Hamiltonian Path Problems to achieve the one-size-fits-all solver. Fang et al. [10] designed a new architecture (INViT) to address the generalization issue, which takes invariant nested views of the instance irrespective of its distribution or problem size.

Our method is motivated by Gao et al. [11], but there are some key differences, which make our policy superior. First and foremost, our local and global features are not treated as independent components. Instead, they are complementary, preventing the local policy from devolving into greedy strategies. In addition, our local policy is integrated within the global encoder, allowing for a seamless fusion of global and local features in the final node embeddings.

3 PROBLEM FORMULATION

We define TSP over a complete graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $v_i \in \mathcal{V}$ represents the node (a.k.a. city or customer), $e_{i,j} \in \mathcal{E}$ represents the edge between v_i and v_j. A tour τ (a.k.a. solution) is defined as a permutation of nodes, which is feasible if and only if it starts from and ends at *depot* while traversing all other nodes exactly once. We focus on the 2D Euclidean TSP in this paper. Given a tour τ, its total cost can be calculated as

$$L(\tau) = cost(e_{\tau_N,\tau_1}) + \sum_{i=1}^{N-1} cost(e_{\tau_i,\tau_{i+1}}) \tag{1}$$

where τ_i denotes the i-th node on the tour, $N = |\mathcal{V}|$ is the nodes number of instance and $cost(e_{\tau_i,\tau_{i+1}}) = \|v_{\tau_i} - v_{\tau_{i+1}}\|_2$ denotes the Euclidean distance between v_{τ_i} and $v_{\tau_{i+1}}$. The objective is to find the optimal tour τ^* with the least total cost among all feasible tours.

A feasible tour can be constructed by sequentially selecting the next node from the rest available nodes. As with many present works, this process can be seen as a Markov Decision Process [24, 25]. The policy $p(\tau|\mathcal{G})$ is defined by the deep neural network to predict the probability of the optimal next node based on the current situation. It is factorized and parameterized by θ as

$$p_\theta(\tau|\mathcal{G}) = \prod_{i=2}^{N-2} p_\theta(\tau_i|\mathcal{G}, \tau_{1:i-1}) \tag{2}$$

where $\mathcal{G}$ denotes a TSP instance and $\tau_{1:i-1}$ is the partial tour before the i-th step. Due to solution symmetry [25], it is equivalent to selecting a random node to start the tour (also as *depot*), so in Equation.(2), i begins with the second step. Additionally, since only one available node left at the $N-1$ step, no prediction is required. This is referred to Neural Construction Method, which constructs the sequential solution in an auto-regressive way [9, 22, 24, 25].

4 MODEL

This section details our G&L-Encoder, which integrates the local policy into the global Transformer encoder, capturing local topological information around each node as a critical complement to global features. The architecture of our entire encoder is illustrated in Figure 1. Following this, an auto-regressive decoder takes the nodes embedding (outputs of the encoder) as inputs to construct the valid tours τ, similar to [24, 25].

4.1 Transformer Encoder (G-Encoder)

Consistent with most present construction methods, we employ the encoder of AM [24] as the cornerstone, called as G-Encoder, which is a primary part of our encoder to capture the global information of the instance by its global graph receptive fields.

Specifically, G-Encoder adopts Transformer without positional embedding [38]. It consists of multiple attention layers (MAL) including multi-head self-attention (MHA), feed-forward layers (FF) and skip-connection [12]. Each MAL takes as input the d_h-dimensional nodes embedding $\left\{h_i^{(l-1)}\right\}_{i=1}^N$, then executes global message passing between the nodes using MHA, as:

$$\hat{h}_i = \text{MHA}^l\left(h_1^{(l-1)}, \ldots, h_N^{(l-1)}\right) \tag{3}$$

where l indicates the index of MAL and these layers do not share parameters. The input of first MAL, $\left\{h_i^{(0)}\right\}_{i=1}^N$, is obtained by a shared linear projection W_{global}, which takes as input each node's coordinates $\{(x_i, y_i)\}_{i=1}^N$.

The node-wise fully connected feed-forward by the FF layer is conducted after the combination of the global and local features (see Section 4.2).

4.2 Local Policy (L-Encoder)

It has been proved that G-Encoder captures global information effectively by aggregating messages from the global scope to each node. However, it lacks a mechanism to take advantage of local topological information surrounding each node. It is significant because the optimal decision for selecting the next node usually comes from the neighborhood of the current node. The deficiency of local information has to make the global policy exhaust itself comprehending or memorizing all details of the instance, potentially contributing to weaker generalization in learnable policies. On the other hand, local features are more stable and independent of global distribution since they contain only the information from their immediate neighborhood. As a result, the possible peculiar nodes within an instance have minimal impact on their distant nodes, enabling local features to better resist global distribution shifts. More analysis can be seen in the Appendix C.

Specifically, we firstly get neighboring nodes for each node using traditional KNN algorithm (set $K = 20$), as $\mathcal{N}(v_i) = \{v'_{i(j)} | j \in \mathcal{N}_K(v_i)\}$. In order to focus on the local topological structures, we use polar coordinates to represent the neighboring nodes of each node, and take itself as the pole of polar coordinates and the x-axis direction as the polar axis.

$$\begin{aligned} v'_{i(j)} &= \left(\widetilde{\rho}_{i(j)}, \widetilde{\rho}_{max_i}, \widetilde{\theta}_{i(j)}\right) \\ \widetilde{\rho}_{i(j)} &= \frac{cost(e_{i,j})}{\widetilde{\rho}_{max_i}} \\ \widetilde{\rho}_{max_i} &= \max(\{cost(e_{i,j}) | j \in \mathcal{N}_K(v_i)\}) \\ \widetilde{\theta}_{i(j)} &= \arctan(v_j - v_i) \end{aligned} \tag{4}$$

where $v'_{i(j)}$ denotes the raw features of node v_j as a neighboring node of v_i. We normalize the radius of neighboring nodes to $[0, 1]$ and add corresponding scale factor $\widetilde{\rho}_{max_i}$ as an additional feature

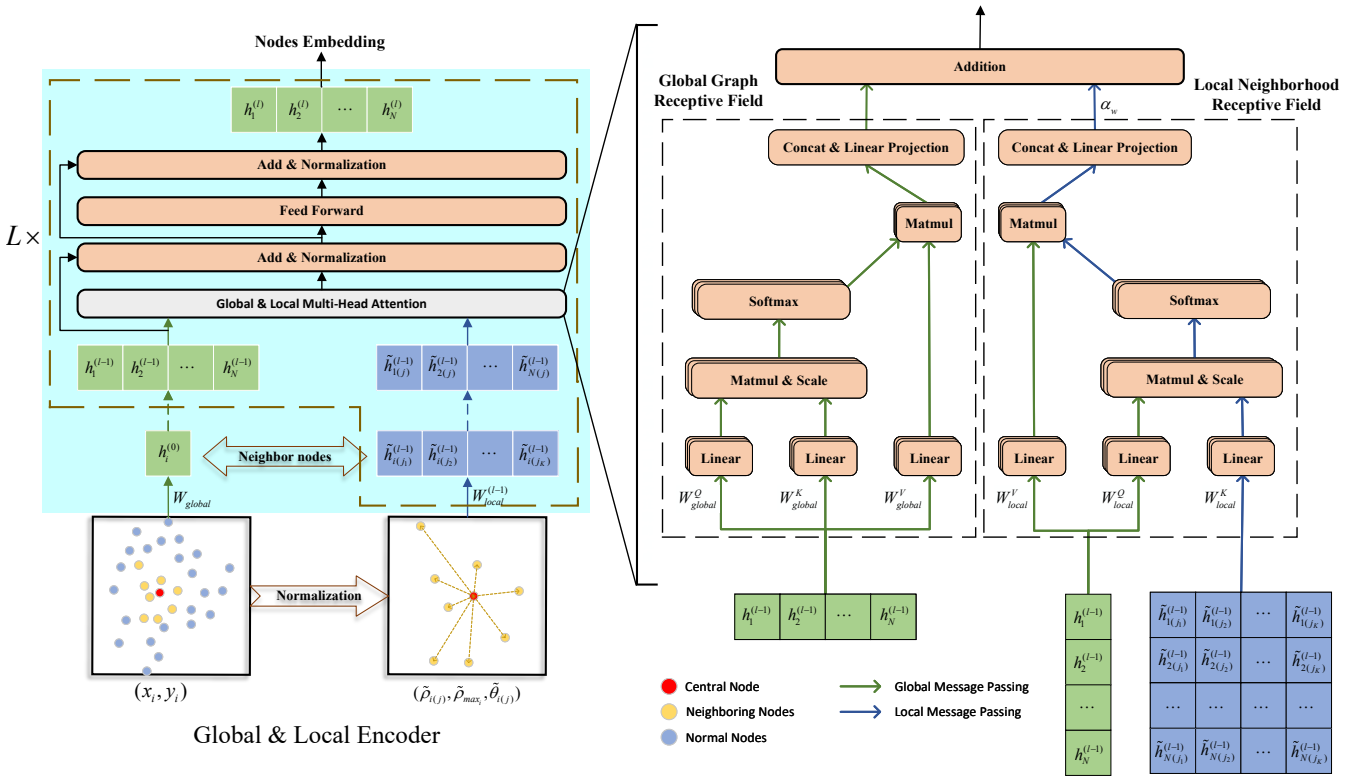

Figure 1: Architecture of the G&L-Encoder, where the L-Encoder takes the local features as keys to aggregate information from neighboring nodes.

to prevent information loss. The neighboring nodes of a single node can be viewed as a small instance consisting of K nodes.

The raw features of neighboring nodes are mapped into d_e-dimension embedding by a shared linear projection $W_{local}^{(l-1)}$, as $\left\{\widetilde{h}_{i(j)}^{(l)} | j \in \mathcal{N}_K(v_i)\right\}_{i=1}^{N}$, whose parameters are not shared among L-Encoder layers. Then, we aggregate the local information from corresponding neighboring nodes using the Multi-Head Attention (MHA). The keys come from the neighboring nodes embedding $\widetilde{h}_{i(j)}^{(l)}$, but the queries and values come from the nodes embedding $h_i^{(l-1)}$, which are the synthesis of global and local features.

$$q_i = W_{local}^{Q} h_i^{(l-1)}, k_{i(j)} = W_{local}^{K} \widetilde{h}_{i(j)}^{(l)}, v_i = W_{local}^{V} h_i^{(l-1)} \tag{5}$$

We compute the compatibility of the queries with their corresponding neighboring nodes, and the vector $\widetilde{h}_i$ that is received by node i is the combination of messages v_j.

$$u_{ij} = \frac{q_i^T k_{i(j)}}{\sqrt{d_e}}, a_{ij} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{ij'}}}, j \in \mathcal{N}_K(v_i) \tag{6}$$

$$\widetilde{h}_i = W_{local}^{C} \sum_j a_{ij} v_j \tag{7}$$

where v_j is in the neighborhood of v_i and $\widetilde{h}_i$ can be regarded as the representation of the local topological information around the v_i.

Before batch normalization (BN) [16] and FF layer, local features are integrated into global features with a learnable factor.

$$\breve{h}_i = \hat{h}_i + \alpha_w \widetilde{h}_i \tag{8}$$

$$h_i^{(l)} = \mathrm{BN}^l(\mathrm{BN}^l(h_i^{(l-1)} + \breve{h}_i) + \mathrm{FF}^l(\breve{h}_i)) \tag{9}$$

4.3 Auto-regressive Decoder

Decoder constructs a solution sequentially based on the node embeddings from the encoder and a special context embedding. The context embedding represents the decoding context, which consists of the embedding of the graph $\bar{h}^{(L)}$, the current node $h_{\tau_{t-1}}^{(L)}$ and the destination node $h_{\tau_1}^{(L)}$:

$$h_{(c)}^{(L)} = f_c\left(\bar{h}^{(L)}, h_{\tau_{t-1}}^{(L)}, h_{\tau_1}^{(L)}\right) \tag{10}$$

where f_c can be viewed as a linear projection to context embedding. During the decoding phase, an obvious intuition is that the demand of visited nodes has been changed, and for TSP, they should no longer influence subsequent decisions. Accordingly, the context embedding need exclude these interference from visited nodes.

Thereby, unlike the POMO, we update the graph embedding after each decision, allowing the model to adapt to the evolving conditions and produce more accurate predictions for subsequent actions.

$$\bar{h}_t^{(L)} = \frac{1}{N-t+1} \sum_{j \notin \tau_{1:t-1}} h_j^{(L)} \tag{11}$$

Algorithm 1 Sampled Steps Supervised Learning

Input: Training set $\mathcal{D} = \{(s, \tau^*)\}$, Multi-start M, Multi-decision T
Output: Model θ

Initialize model's params θ
for $e = 1$ **to** E **do**
  Set problem size: N
  $s_i \sim SampleData(\mathcal{D}, N)$ **for** $i \in \{1, \ldots, B\}$ ▷ Data batch
  $s_{i,j} \sim SampleTrajectories(s_i)$ **for** $j \in \{1, \ldots, M\}$ ▷ Sample M trajectories for each instance
  $\mathcal{T} \leftarrow SampleSteps(T, N)$ ▷ Pick T decision steps
  for t **in** $\mathcal{T}$ **do**
    $\mathcal{P}_{i,j,t} \leftarrow p_\theta(s_{i,j}, \tau^*_{1:t-1})$
  end for
  $\mathcal{L}_\theta \leftarrow \frac{1}{B*M} \sum_{i,j} \sum_{t \in \mathcal{T}} H(\mathcal{P}_{i,j,t}, \mathcal{P}^*_{i,j,t})$ ▷ The function H indicates cross entropy
  $\theta \leftarrow Adam(\theta, \nabla_\theta \mathcal{L}_\theta)$
end for

where t indicates the t-th decision step and $\tau_{1:t-1}$ is corresponding partial solution. The prediction of next node is computed as follows:

$$\hat{h}_{(c)} = MHA_c\left(\left\{h_i^{(L)}\right\}_{i=1}^{N}, h_{(c)}^{(L)}\right) \tag{12}$$

$$u_1, \ldots, u_N = SHA_c\left(\left\{h_i^{(L)}\right\}_{i=1}^{N}, \hat{h}_{(c)}\right) \tag{13}$$

$$p_\theta(\tau_t = i | \mathcal{G}, \tau_{1:t-1}) = \frac{e^{u_i}}{\sum_j e^{u_j}} \tag{14}$$

where the decoder computes a multi-head attention on top of the encoder, but with messages only to the context embedding. Then, the final probabilities of optimal next node are computed using a single head attention with clipping and the unsatisfied nodes masked. The detailed structure of MHA_c and SHA_c can be found in [24, 25].

5 SAMPLED STEPS SUPERVISED LEARNING

In this section, we present our training framework based on the supervised learning. The optimal solutions of COPs are selected from exponential discrete space, which cover abundant information bound to the heuristic strategies and node distribution characteristics (local and global). A desired supervised learning method should be capable of rationally uncovering this implicit information and efficiently guiding the model to converge on a generalized policy, rather than overfitting to specific distributions [21].

We design Sampled Steps Supervised Learning (SSSL) which estimates the gradients of model parameters by predicting several sampled steps instead of all steps during the decoding phase. The fundamental principle is that the complete optimal tour is compounded from a series of optimal decision steps. Thus, it is equivalent for training to learn from the complete tour or several sampled steps in the supervised framework, the latter can save lots of computational resources and significantly expand the limited training labels.

Algorithm 1 presents the SSSL training framework with a mini-batch. Inspired by [25], we firstly choose M trajectories with different starting nodes and exploratory directions from optimal tour τ^*. Then, randomly pick T different decision steps $\mathcal{T} = \{t\}_T$ (a.k.a. lengths of partial tours) to predict. The model computes the probabilities $p_\theta(\tau_t | s, \tau^*_{1:t-1})$ for available nodes. The cross-entropy loss is employed to measure the gap between the prediction and optimal distribution. We minimize the sum of the cross-entropy loss of different decisions on the same trajectory over mini-batches.

$$\mathcal{L}_\theta = \mathbb{E}_{s,\tau^* \sim \mathcal{D}} \mathbb{E}_{t \sim \tau^*} \sum_{t \in \mathcal{T}} H(p_\theta(\tau_t = \tau^*_t | s, \tau^*_{1:t-1}), \mathbf{1}) \tag{15}$$

The computational complexity analysis, an improved SSSL, and more discussion can be seen in Appendix A.6, B.2.1, and B.4.2.

6 EXPERIMENTS

In this section, we empirically evaluate the effectiveness of our method on synthetic and benchmark of TSP with various sizes and distributions. And we intentionally compare the effects of training distributions on the final model's generalization.

Problem setting We generate 3 different training sets, respectively termed U, UCM, UCM-CSize. The U follows the standard data generation procedure of the previous work [24] with uniform distribution and fixed problem size $N = 100$. The UCM follows the generation method of [2] including uniform, cluster and mixed distributions with fixed size $N = 100$. The UCM-CSize includes the same distributions as the UCM, but various problem sizes $N \in \{69, 83, 100, 120, 144\}$. These training sets consist of 1M instances for training respectively.

In order to sufficiently evaluate the model performance under the cross-size and cross-distribution conditions, we design 24 test tasks with diverse sizes $N \in \{100, 150, 200, 300\}$ and six node distributions, i.e. uniform, cluster, mixed [2], explosion, expansion, rotation [3], while each test task includes 1K instances.

The optimal solutions of all datasets are obtained by Concorde [8] designed specifically for TSP.

Baselines For the non-learning solver, we take the results from Concorde as the optimal. For learning-based solver, we choose to compare with representative construction methods, which is convenient to set the same sampling strategy, including POMO [25], Sym-NCO [22], ELG [11], AMDKD-POMO [2], Omini-POMO [48] and INViT-3V [10].

Unless otherwise stated, all learning-based solvers perform inference using multi-start greedy rollout with ×8 instance augmentations sampling proposed by [25] on a single NVIDIA RTX 3090 during the evaluation. It is fairly that we can focus on contrasting models' performance. For the comparison methods, we use the pre-trained models published by their authors. We report the average gap to the optimal solutions and the total inference time on each test task.

Hyper-parameters All our models are trained for 2010 epochs and every epoch processes 100K instances with $B = 64$ batch size and 2 augment factor (about 1563 batches in each epoch). We empirically set $M = 100$ and $T = 3$ regardless of the problem sizes. The dimension of local embedding d_e is set to 32 and the others keep the same with [25]. Each training epoch takes about 2.0 minutes

Table 1: Empirical results on all test tasks with various distributions and problem sizes. Each task includes 1K instances.

	TSP-100		TSP-150		TSP-200		TSP-300		TSP-100		TSP-150		TSP-200		TSP-300	
	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
	Uniform								**Cluster**							
Concorde	0.00%	4.4m	0.00%	12m	0.00%	23m	0.00%	1.2h	0.00%	5.0m	0.00%	12m	0.00%	20m	0.00%	1h
POMO	0.15%	6.1s	0.58%	19s	1.60%	37s	6.15%	2.0m	1.72%	5.6s	3.41%	19s	6.12%	37s	12.97%	2.0m
Sym-NCO	0.15%	6.2s	0.55%	19s	1.60%	37s	6.52%	2.0m	1.70%	5.7s	3.10%	19s	5.56%	37s	12.44%	2.0m
INViT-V3	1.23%	-	1.91%	-	2.46%	-	3.13%	-	1.79%	-	2.53%	-	3.06%	-	3.86%	-
ELG	0.24%	17s	0.71%	50s	1.50%	1.7m	3.48%	4.9m	1.33%	17s	2.35%	49s	3.51%	1.7m	5.73%	4.8m
Our(U)	**0.01%**	7.0s	0.12%	21s	0.83%	40s	4.94%	2.0m	0.41%	6.6s	1.37%	21s	3.37%	41s	8.90%	2.1m
AMDKD-POMO	0.36%	4.8s	1.16%	17s	2.82%	33s	7.83%	1.8m	0.38%	4.8s	1.24%	17s	2.98%	33s	8.17%	1.8m
ELG#	0.31%	18s	0.84%	51s	1.68%	1.7m	3.61%	5.0m	0.41%	18s	1.08%	51s	1.98%	1.7m	3.98%	5.0m
Our(UCM)	0.01%	7.0s	0.15%	21s	0.78%	40s	4.79%	2.1m	**0.03%**	6.6s	0.25%	21s	1.08%	40s	5.07%	2.1m
Omni-POMO	1.26%	5.5s	1.52%	19s	1.95%	37s	3.23%	2.0m	1.38%	5.5s	1.73%	19s	2.30%	37s	3.93%	2.0m
ELG##	0.41%	18s	0.93%	51s	1.62%	1.7m	3.08%	5.0m	0.53%	18s	1.22%	51s	1.96%	1.7m	3.52%	5.0m
Our(UCM-CSize)	0.01%	7.0s	**0.09%**	20s	**0.27%**	40s	**0.99%**	2.1m	0.03%	6.5s	**0.15%**	21s	**0.45%**	40s	**1.63%**	2.1m
	Mixed								**Explosion**							
Concorde	0.00%	5.0m	0.00%	11m	0.00%	21m	0.00%	52m	0.00%	4.2m	0.00%	9.8m	0.00%	21m	0.00%	58m
POMO	0.84%	5.6s	2.16%	19s	4.11%	37s	9.15%	2.0m	0.19%	5.6s	0.77%	19s	2.29%	37s	7.62%	2.0m
Sym-NCO	0.71%	5.7s	1.97%	19s	3.92%	37s	9.25%	2.0m	0.14%	5.7s	0.62%	19s	2.03%	37s	7.61%	2.0m
INViT-V3	1.66%	-	2.34%	-	2.79%	-	3.36%	-	1.19%	-	2.06%	-	2.69%	-	3.68%	-
ELG	0.96%	17s	1.82%	50s	2.75%	1.7m	4.71%	4.9m	0.22%	17s	0.73%	50s	1.65%	1.7m	3.71%	4.8m
Our(U)	0.21%	7.0s	0.83%	21s	2.04%	40s	6.10%	2.1m	0.01%	6.4s	0.17%	20s	1.07%	40s	5.87%	2.1m
AMDKD-POMO	0.41%	4.8s	1.17%	17s	2.63%	33s	7.23%	1.8m	0.28%	4.8s	1.01%	17s	2.65%	33s	7.66%	1.8m
ELG#	0.41%	18s	0.97%	51s	1.72%	1.7m	3.49%	5.0m	0.20%	18s	0.68%	51s	1.52%	1.7m	3.48%	5.0m
Our(UCM)	**0.03%**	6.6s	0.21%	21s	0.81%	40s	4.20%	2.2m	**0.01%**	6.6s	0.14%	21s	0.84%	40s	5.53%	2.1m
Omni-POMO	2.19%	5.5s	2.69%	19s	3.15%	37s	4.63%	2.0m	0.96%	5.5s	1.29%	19s	1.76%	37s	3.17%	2.0m
ELG##	0.54%	18s	1.10%	51s	1.74%	1.7m	3.05%	5.0m	0.28%	18s	0.79%	51s	1.56%	1.7m	3.14%	5.0m
Our(UCM -CSize)	0.04%	6.5s	**0.17%**	21s	**0.39%**	40s	**1.16%**	2.1m	0.01%	6.5s	**0.09%**	21s	**0.31%**	40s	**1.42%**	2.1m
	Expansion								**Rotation**							
Concorde	0.00%	8.0m	0.00%	17m	0.00%	29m	0.00%	1.1h	0.00%	3.9m	0.00%	9.6m	0.00%	21m	0.00%	1h
POMO	0.58%	5.6s	1.87%	19s	4.17%	37s	10.31%	2.0m	0.55%	5.6s	1.33%	19s	3.02%	37s	8.61%	2.0m
Sym-NCO	0.50%	5.7s	1.67%	19s	3.91%	37s	10.40%	2.0m	0.54%	5.7s	1.20%	19s	2.77%	37s	8.68%	2.0m
INViT-V3	2.28%	-	3.28%	-	3.87%	-	4.75%	-	1.25%	-	1.96%	-	2.44%	-	3.12%	-
ELG	0.60%	17s	1.50%	49s	2.61%	1.7m	4.73%	4.8m	0.52%	17s	1.14%	49s	2.11%	1.7m	4.16%	4.9m
Our(U)	0.34%	6.6s	1.41%	21s	3.68%	40s	9.72%	2.1m	0.03%	6.6s	0.27%	21s	1.34%	40s	6.17%	2.1m
AMDKD-POMO	0.64%	4.8s	1.81%	17s	3.72%	33s	8.79%	1.8m	0.30%	4.8s	0.99%	17s	2.60%	33s	7.50%	1.8m
ELG#	0.51%	18s	1.23%	51s	2.10%	1.7m	4.01%	5.0m	0.31%	18s	0.86%	51s	1.75%	1.7m	3.75%	5.0m
Our(UCM)	0.18%	6.6s	0.73%	21s	2.06%	40s	7.28%	2.2m	0.02%	6.6s	0.15%	21s	0.83%	40s	4.93%	2.1m
Omni-POMO	1.43%	5.5s	1.79%	19s	2.24%	37s	3.67%	2.0m	1.22%	5.5s	1.53%	19s	2.04%	37s	3.54%	2.0m
ELG##	0.28%	18s	0.79%	51s	1.56%	1.7m	3.14%	5.0m	0.42%	18s	1.01%	51s	1.75%	1.7m	3.34%	5.0m
Our(UCM -CSize)	**0.15%**	6.5s	**0.58%**	21s	**1.17%**	40s	**2.79%**	2.1m	**0.02%**	6.5s	**0.09%**	21s	**0.29%**	40s	**1.15%**	2.1m

on a single NVIDIA RTX 3090 (about 3 days for the entire training, compared with 2 weeks on NVIDIA A100 GPU of the original method [22]). More detailed settings can be found in Appendix A.

Performance Metrics For each comparison method, we report the average gap to the optimal solution from Concorde and the total inference time on each test task. The gap corresponding to a instance s is calculated as follows [10]:

$$gap = \frac{\mathcal{D}_s(F^{model}) - \mathcal{D}_s(F^{opt})}{\mathcal{D}_s(F^{opt})} \times 100\% \tag{16}$$

where $\mathcal{D}_s(F^{model})$ represents the length of solutions from neural models, and $\mathcal{D}_s(F^{opt})$ represents the length of the optimal solutions.

6.1 Results on Synthetic Instances

Table 1 demonstrates the results on all test tasks and partial results for some non-learning heuristics are attached in the Appendix B.1 due to the limited space. We group the results of the neural methods into three groups according to their training distributions (stated on their papers). The POMO, Sym-NCO, INViT-V3 and ELG are trained on uniform. The AMDKD-POMO is trained on three distributions, including uniform, cluster, mixed, which is the same as UCM. Both above keep the problem size fixed $N = 100$. The Omni-POMO is trained on more complex distributions including uniform and gaussian mixture with diverse sizes $N \in [50, 200]$. We retrain the ELG using the instance generator of UCM and UCM-CSize, denoted as *ELG#* and *ELG##*. All neural methods, except INViT-V3, utilize multi-start rollout sampling with ×8 instance augmentation. INViT-V3 employs ×128 instance augmentation in place of multi-start rollout sampling, and its inference time cannot be directly compared to other methods as it does not implement parallelized evaluation.

In the following, we conduct a detailed analysis of the models' performance on synthetic test tasks.

Generalization issue is still a significant challenge. As shown in Table 1, the POMO and Sym-NCO suffer from huge degeneration under cross-distribution and cross-size conditions, particularly with cluster distributions. Creating more complex training environments has proven beneficial in helping models acquire generalizable knowledge, as evidenced by the substantial improvements of AMDKD-POMO and Omni-POMO over POMO. However, an intriguing phenomenon should be noted: models trained in such complex environments fail to demonstrate reasonable performance on some small and simple test tasks, e.g. TSP-100(Uniform, Explosion, Expansion). Notably, Omni-POMO exhibits significant performance deterioration on TSP-100 and TSP-150 tasks compared to POMO. This raises legitimate concerns about whether the current model architectures or the information they encode are sufficient to effectively support stronger generalization capabilities.

Local topological information improve the learning capabilities of models. Unlike POMO and Sym-NCO, the INViT-V3, ELG and our models all incorporate local topological information, where INViT-V3 removes global features to accommodate large-scale instances. We retrain the ELG using the instance generator of UCM and UCM-CSize, separately. As shown in Table 1, when the training distribution is consistent, both ELG and our models outperform AMDKD-POMO and Omni-POMO, even without introducing additional training techniques, like meta-learning. This demonstrates that incorporating local topological information can effectively enhance the model's learning capabilities.

Table 2: Optimality gap on TSPLIB ($1 \le N \le 500$).

TSPLIB	1 ~ 100	101 ~ 300	300 ~ 500
POMO	1.16%	4.44%	17.24%
Sym-NCO	1.04%	3.82%	19.84%
INViT-V3	1.14%	3.26%	5.85%
ELG	0.55%	1.89%	6.00%
Our(U)	0.30%	2.93%	15.84%
AMDKD-POMO	0.45%	2.89%	12.17%
ELG#	0.46%	1.42%	5.34%
Our(UCM)	**0.27%**	2.48%	14.63%
Sym-NCO*	0.34%	2.37%	11.88%
Omni-POMO	1.66%	1.99%	6.65%
ELG##	0.45%	1.49%	**4.80%**
Our(UCM-CSize)	0.27%	**1.38%**	7.76%

Training distributions show substantial effects on generalization. We constructed three progressively complex training distributions, as U, UCM and UCM-CSize, to examine how training distributions impact model performance. In both U and UCM, the problem size is fixed at $N = 100$, with UCM featuring more diverse node distributions. As shown in Table 1, *ELG#* achieves notable improvements over ELG in most tasks, except for the Uniform distribution. Our(U) and Our(UCM) also exhibit similar phenomena; however, in the Uniform task, Our(UCM) achieves performance comparable to that of Our(U), with a partial enhancement in cross-size capabilities. The UCM-CSize introduces instances of varying sizes, enabling the model to learn more cross-size knowledge. With the exception of the Expansion distribution, *ELG##* demonstrates an overall decline in performance compared to *ELG#* on the TSP100-150 tasks, while exhibiting improvements on the TSP200-300 tasks. Unlike *ELG##*, Our(UCM-CSize) exhibits enhanced cross-size capabilities compared to Our(UCM), while simultaneously retaining the original performance for smaller-scale instances. By comparing the performance of Our(U), Our(UCM) and Our(UCM-CSize) across various test tasks, it can be concluded that a rich and complex training environment is essential for model training, as it effectively mitigates generalization issues. Meanwhile, Our(UCM-CSize) significantly surpasses Omni-POMO, even though the latter was trained in a more complex environment than UCM-CSize. This indicates that model architectures focused solely on global features are insufficient for effectively addressing distribution shifts, further emphasizing the critical role of local topological information in enhancing model generalization.

The performance valuation on large scale instances is provided in the Appendix B.5.

6.2 Results on TSPLIB Benchmark

We evaluate the generalization of our method on TSPLIB [35], which contains various instances of unknown distributions and problem

Table 3: Ablation studies. All models are trained on the UCM-CSize using SSSL framework. The results of all test tasks are grouped into In-Distributions (ID), cross-distributions (CD), cross-sizes (CS) and cross-distributions&cross-sizes (CD&CS).

Method	G_{ID}	G_{OoD}		
		G_{CD}	G_{CS}	$G_{CD\&CS}$
Sym-NCO*	0.23%	0.33%	1.56%	2.28%
Our (*w/o L-Encoder*)	0.21%	0.33%	1.50%	2.28%
Our (*w/o new graph embedding*)	0.10%	0.20%	0.92%	1.37%
Our	**0.08%**	**0.16%**	**0.82%**	**1.19%**

sizes. As shown in Table 2, our models also show better generalization on medium-sized instances.

However, it should be clarified that our method exhibits anomalous performance degradation on certain instances compared to similar approaches, specifically with gaps reaching 2.19%, 8.39%, 8.38% and 25.73% on *pr144.tsp*, *ts225.tsp*, *pr226.tsp* and *fl417.tsp* instances, respectively, significantly exceeding the average gap for their corresponding scales. For the other instances, our method achieves optimal or near-optimal results among comparable methods. Detailed experimental results and more discussion can be found in Appendix B.2.

6.3 Ablation Studies

In this section, we perform ablation studies to evaluate the impact of various components within our method. Since our model builds upon the Sym-NCO [22], we utilize it as a fair baseline for comparison. Meanwhile, we retrain Sym-NCO on UCM-CSize using the SSSL framework, denoted as *Sym-NCO**, to eliminate the potential influence of training scheme and training distributions. As illustrated in Table 3, we divide all test tasks into four groups: In-Distributions (ID) includes TSP-100,150(Uniform, Cluster, Mixed), Cross-Distributions (CD) includes TSP-100,150(Explosion, Expansion, Rotation), Cross-Size (CS) includes TSP-200,300(Uniform, Cluster, Mixed) and CD&CS includes TSP-200,300(Explosion, Expansion, Rotation). The results indicate that the *L-Encoder* significantly enhances model performance across all task categories, with improvements of 56.52%, 39.39%, 41.03% and 39.91% in ID, CD, CS and CD&CS, respectively. The *L-Encoder* serves as the primary contributor to the performance enhancement, while the *new graph embedding* offers comparatively modest improvements.

We examine the influence of the local neighboring node size K. The results show that when the K is in a reasonable range, it does not affect the performance significantly, aligning with the results of [11]. Detailed results are provided in Appendix B.4.1.

6.4 Training Curve of α_w

The coefficient α_w are the d_h-dimensional parameters within each multi-attention layer (MAL) that adjusts the relative contributions of global and local information. A larger α_w value for a given dimension indicates that the node embeddings output by the MAL in that dimension contain a greater amount of local information. We initialize α_w using *torch.nn.init.ones_* function. Figure 2 depicts the evolution of the mean and variance of α_w across their dimensions during training. The overall downward trend suggests an improvement in the model's capacity to capture global information. In the later stages of training, most of the α_w values gradually converge to values greater than 0.4, indicating that local information continues to be indispensable in the problem-solving process. A more detailed analysis is provided in the Appendix B.3.

We visualize the impact of the L-Encoder on the node embeddings and further discuss its role in alleviating the issue of over-smoothing in node embeddings. Detailed analysis is provided in Appendix B.6.

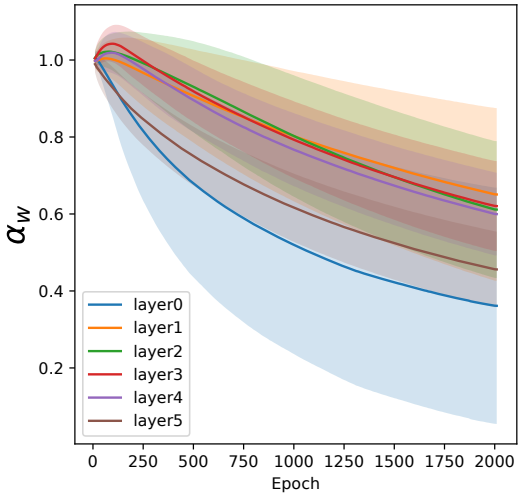

Figure 2: Training curve of α_w, which illustrates the evolution of the mean and variance of each α_w across their dimensions during the training of *Our(UCM-CSize)*.

7 CONCLUSION

This paper studies the impact of local topological information on the generalization of neural methods for solving TSP. We introduce the G&L-Encoder, a novel architecture that integrates global and local information by embedding our local policies within a global Transformer-based encoder, enabling the model to capture local topological information as an essential complement to global features. In order to efficiently exploit the implicit knowledge inherent in high-quality solutions, we design Sampled Steps Supervised Learning, which estimates the gradients by sampling a subset of decision steps, eliminating the need to construct complete tours during training. To evaluate the effectiveness of our method, we conduct comprehensive experiments on synthetic and real-world instances of the TSP with various distributions and sizes. The experimental results demonstrate that our method significantly improves the generalization against baselines and achieves state-of-the-art results on medium-scale TSP tasks. We hope our work can inspire further research towards building the truly generic policies for routing problems.

REFERENCES

[1] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

[2] Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. 2022. Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems* 35 (2022), 31226–31238.

[3] Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. 2019. Evolving diverse TSP instances by means of novel and creative mutation operators. In *Proceedings of the 15th ACM/SIGEVO conference on foundations of genetic algorithms*. 58–71.

[4] Xavier Bresson and Thomas Laurent. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553* (2017).

[5] Xinyun Chen and Yuandong Tian. 2019. Learning to perform local rewriting for combinatorial optimization. *Advances in neural information processing systems* 32 (2019).

[6] William J Cook, David L Applegate, Robert E Bixby, and Vasek Chvatal. 2011. *The traveling salesman problem: a computational study*. Princeton university press.

[7] Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. 2020. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian conference on machine learning*. PMLR, 465–480.

[8] Applegate David, Bixby Robert, Chvatal Vasek, and Cool William. 2006. Concorde TSP Solver. (2006).

[9] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. 2018. Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*. Springer, 170–181.

[10] Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. 2024. INViT: A Generalizable Routing Problem Solver with Invariant Nested View Transformer. *arXiv preprint arXiv:2402.02317* (2024).

[11] Chengrui Gao, Haopu Shang, Ke Xue, Dong Li, and Chao Qian. 2023. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. *arXiv preprint arXiv:2308.14104* (2023).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Keld Helsgaun. 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European journal of operational research* 126, 1 (2000), 106–130.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[15] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. 2021. Graph neural network guided local search for the traveling salesperson problem. *arXiv preprint arXiv:2110.05291* (2021).

[16] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 448–456. https://proceedings.mlr.press/v37/ioffe15.html

[17] Yuan Jiang, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. 2023. Multi-view graph contrastive learning for solving vehicle routing problems. In *Uncertainty in Artificial Intelligence*. PMLR, 984–994.

[18] Yuan Jiang, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. 2022. Learning to solve routing problems via distributionally robust optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9786–9794.

[19] Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. 2023. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 8132–8140.

[20] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. 2020. Learning the travelling salesperson problem requires rethinking generalization. *arXiv preprint arXiv:2006.07054* (2020).

[21] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227* (2019).

[22] Minsu Kim, Junyoung Park, and Jinkyoo Park. 2022. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems* 35 (2022), 1936–1949.

[23] Minjun Kim, Junyoung Park, and Jinkyoo Park. 2023. Learning to CROSS exchange to solve min-max vehicle routing problems. In *The Eleventh International Conference on Learning Representations*.

[24] Wouter Kool, Herke Van Hoof, and Max Welling. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475* (2018).

[25] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 21188–21198.

[26] Yixuan Li, Can Chen, Jiajun Li, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, and Wanyuan Wang. 2024. Fast and Interpretable Mixed-Integer Linear Program Solving by Learning Model Reduction. *arXiv preprint arXiv:2501.00307* (2024).

[27] Yixuan Li, Wanyuan Wang, Weiyi Xu, Yanchen Deng, and Weiwei Wu. 2024. Factor Graph Neural Network Meets Max-Sum: A Real-Time Route Planning Algorithm for Massive-Scale Trips. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. 1165–1173.

[28] Shengcai Liu, Yu Zhang, Ke Tang, and Xin Yao. 2023. How good is neural combinatorial optimization? A systematic evaluation on the traveling salesman problem. *IEEE Computational Intelligence Magazine* 18, 3 (2023), 14–28.

[29] Hao Lu, Xingwen Zhang, and Shuang Yang. 2019. A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*.

[30] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. 2023. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems* 36 (2023), 8845–8864.

[31] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. 2019. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936* (2019).

[32] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems* 34 (2021), 11096–11107.

[33] Sahil Manchanda, Sofia Michel, Darko Drakulic, and Jean-Marc Andreoli. 2022. On the generalization of neural combinatorial optimization heuristics. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 426–442.

[34] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. 2022. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems* 35 (2022), 25531–25546.

[35] Gerhard Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.

[36] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. 2019. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731* (2019).

[37] Zhiqing Sun and Yiming Yang. 2023. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems* 36 (2023), 3706–3731.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[39] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. *Advances in neural information processing systems* 28 (2015).

[40] Chenguang Wang, Yaodong Yang, Oliver Slumbers, Congying Han, Tiande Guo, Haifeng Zhang, and Jun Wang. 2021. A game-theoretic approach for improving generalization ability of TSP solvers. *arXiv preprint arXiv:2110.15105* (2021).

[41] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8 (1992), 229–256.

[42] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2021. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems* 33, 9 (2021), 5057–5069.

[43] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. 2021. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 12042–12049.

[44] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. 2022. Generative adversarial training for neural combinatorial optimization models. (2022).

[45] Hua Yang, Minghao Zhao, Lei Yuan, Yang Yu, Zhenhua Li, and Ming Gu. 2023. Memory-efficient transformer-based network model for traveling salesman problem. *Neural Networks* 161 (2023), 589–597.

[46] Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. 2024. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 20284–20292.

[47] Zeyang Zhang, Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2022. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9136–9144.

[48] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. 2023. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*. PMLR, 42769–42789.