EFormer: An Effective Edge-based Transformer for Vehicle Routing Problems

Dian Meng[1,4] , **Zhiguang Cao**[2] , **Yaoxin Wu** [3] , **Yaqing Hou**[1,4*] , **Hongwei Ge**[1,4] and **Qiang Zhang**[1,4]
[1]School of Computer Science and Technology, Dalian University of Technology (DUT)
[2]School of Computing and Information Systems, Singapore Management University
[3]Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology
[4]Key Laboratory of Social Computing and Cognitive Intelligence (DUT), Ministry of Education, China
mengdian@mail.dlut.edu.cn, zhiguangcao@outlook.com, wyxacc@hotmail.com, {houyq, gehw, zhangq}@dlut.edu.cn

Abstract

Recent neural heuristics for the Vehicle Routing Problem (VRP) primarily rely on node coordinates as input, which may be less effective in practical scenarios where real cost metrics—such as edge-based distances—are more relevant. To address this limitation, we introduce EFormer, an Edge-based Transformer model that uses edge as the sole input for VRPs. Our approach employs a precoder module with a mixed-score attention mechanism to convert edge information into temporary node embeddings. We also present a parallel encoding strategy characterized by a graph encoder and a node encoder, each responsible for processing graph and node embeddings in distinct feature spaces, respectively. This design yields a more comprehensive representation of the global relationships among edges. In the decoding phase, parallel context embedding and multi-query integration are used to compute separate attention mechanisms over the two encoded embeddings, facilitating efficient path construction. We train EFormer using reinforcement learning in an autoregressive manner. Extensive experiments on the Traveling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP) reveal that EFormer outperforms established baselines on synthetic datasets, including large-scale and diverse distributions. Moreover, EFormer demonstrates strong generalization on real-world instances from TSPLib and CVRPLib. These findings confirm the effectiveness of EFormer's core design in solving VRPs.

1 Introduction

The vehicle routing problem (VRP), a fundamental NP-hard combinatorial optimization problem (COP), appears in numerous real-world contexts, including logistics [Konstantakopoulos *et al.*, 2022], navigation systems [Elgarej *et al.*, 2021], and circuit design [Brophy and Voigt, 2014]. Despite extensive research across various fields, VRPs remain notably challenging due to their inherent computational complexity [Ausiello *et al.*, 2012]. Approaches to solving VRPs can be divided into exact algorithms and heuristic algorithms [Helsgaun, 2017]. Exact algorithms, although theoretically robust, often face scalability issues when applied to large instances because of their high computational demands. In contrast, heuristic algorithms are generally more practical yet depend heavily on manually crafted rules and domain-specific knowledge, which restricts their applicability and generalizability.

Recently, there has been a surge in neural heuristics that leverage deep (reinforcement) learning to solve VRPs. These methods learn problem-solving strategies end-to-end from data, offering a novel and efficient perspective on VRPs [Kool *et al.*, 2019]. Compared to traditional heuristics, neural heuristics typically provide higher solution efficiency and stronger generalization capabilities [Zhang *et al.*, 2025].

Nevertheless, most existing neural heuristics rely heavily on node coordinates (or their embeddings) as a crucial input to the model, with many methods generating problem instances from these coordinates to train neural networks. The core assumption is that the relationship between coordinates and distance can be readily learned by the network when searching for the shortest path, particularly in classical Euclidean spaces. In these approaches, each iteration selects a node from the problem instance, and the encoded information of the remaining nodes is then used to incrementally construct the solution by inferring the corresponding Euclidean distance. However, this prevalent focus on node coordinates often lacks the robustness and generalizability required in practical applications. When distances cannot be easily inferred from coordinates alone, such methods tend to struggle, reducing their effectiveness in solving VRPs—especially in scenarios where the input space deviates from idealized conditions.

To address this issue, we propose a neural heuristic featuring an Edge-based Transformer (EFormer) model that exclusively relies on edge information as the original input. Specifically, we introduce a precoder module that employs a multi-head mixed-score attention mechanism to convert edge information into temporary node embeddings. We then design a parallel encoding strategy comprising two encoders: a graph encoder and a node encoder. The graph encoder employs a residual gated graph convolution network (GCN) to process sparse graph embeddings, while the node encoder leverages attention mechanisms to independently process temporary node embeddings. By encoding these two embedding types in sepa-

[*]Yaqing Hou is the Corresponding author.

rate feature spaces, the model achieves a more comprehensive representation of global edge relationships. Finally, parallel context embedding and multiple-query integration are used to decode the two types of encoded features, facilitating the effective construction of a complete path. The EFormer is trained using reinforcement learning in an autoregressive manner and achieves favorable performance on the TSP and CVRP. Accordingly, our contributions can be summarized as follows:

- **Novel Edge-based Transformer (EFormer).** We propose a new and practical edge-based Transformer model designed to leverage the edge information for solving VRPs. The introduction of a multi-head mixed-score attention mechanism enables the extraction of temporary node features directly from edge weights.
- **Parallel Encoding Strategy.** We develop a parallel encoding strategy that utilizes residual gated graph convolution networks to encode graph embeddings and attention mechanisms to process node embeddings. By encoding graph and node embeddings in separate feature spaces, the model generates more comprehensive global edge relationship embeddings. A multiple-query integration method is then used to decode the embeddings, facilitating complete path construction.
- **Favorable Performance and Versatility.** Our purely edge-based method demonstrates strong performance on TSP and CVRP, surpassing other edge-based neural methods. Its robust generalization is evident in its performance on real-world instances from TSPLib and CVRPLib across diverse scales and distributions. The edge-based nature of EFormer also allows us to apply it to Asymmetric TSP (Appendix D). Furthermore, we demonstrate its versatility by adapting it to solve VRPs using only node information as input (Appendix C)[1].

2 Related Work

2.1 Node-based Neural Heuristics for VRP

Graph neural network-based methods. Graph neural networks (GNNs) [Scarselli *et al.*, 2008] provide a flexible framework for learning message-passing strategies among nodes, making them applicable to graphs of arbitrary size. In the context of routing problems, GNNs typically predict edge probabilities or scores, which are then leveraged by search algorithms (e.g., beam search, tree search, or guided local search) to produce approximate solutions [Khalil *et al.*, 2017; Li *et al.*, 2018; Nowak *et al.*, 2017; Joshi *et al.*, 2019; Fu *et al.*, 2021; Xin *et al.*, 2021; Hudson *et al.*, 2021; Kool *et al.*, 2022; Min *et al.*, 2024]. [Khalil *et al.*, 2017] proposed one of the earliest unified frameworks that combine reinforcement learning with graph embeddings to solve various combinatorial optimization (CO) problems on graphs. In another line of work, [Li *et al.*, 2018] incorporated advanced Graph Convolutional Networks (GCNs) [Kipf and Welling, 2016] alongside tree search to explore the solution space for CO problems. [Nowak *et al.*, 2017] applied supervised learning to train a GNN and employed beam search to obtain feasible solutions. [Joshi *et al.*, 2019] designed a GCN model to predict a "heatmap" of edge probabilities in TSP instances, which guides beam search to produce feasible solutions.

[1]http://arxiv.org/abs/2506.16428

Transformer-based constructive methods. Among various neural construction heuristics, the Transformer [Vaswani, 2017] represents a major breakthrough and has progressively become the leading approach for solving VRPs. These methods typically construct solutions incrementally by selecting one node at a time from the problem instance. Representative methods in this category include [Nazari *et al.*, 2018; Kim *et al.*, 2021; Kool *et al.*, 2019; Kwon *et al.*, 2020; Jin *et al.*, 2023b; Drakulic *et al.*, 2023; Luo *et al.*, 2023; Huang *et al.*, 2025; Lin *et al.*, 2024]. Specifically, [Kool *et al.*, 2019] is the first to leverage the Transformer architecture in a method called Attention Model (AM), thereby introducing a more powerful neural heuristic for VRPs. One notable variant of AM is POMO [Kwon *et al.*, 2020], which applies multiple optimal policies to significantly enhance AM's learning and inference capabilities. More recently, [Luo *et al.*, 2023] proposed the Light Encoder and Heavy Decoder (LEHD) model, trained via supervised learning on 100-node instances. LEHD not only achieves higher-quality solutions but also demonstrates strong generalization capabilities.

Transformer-based improvement methods. Neural improvement heuristics iteratively refine an initial feasible solution until a stopping criterion (e.g., convergence) is reached. Drawing inspiration from classical local search algorithms, they can optimize sub-problems or apply improvement operators (e.g., k-opt) to enhance solution quality. Representative improvement-based approaches include [Lu *et al.*, 2019; Barrett *et al.*, 2020; Wu *et al.*, 2021; Ma *et al.*, 2021; Li *et al.*, 2021; Wang *et al.*, 2021; Kim *et al.*, 2023; Cheng *et al.*, 2023].

2.2 Edge-based Neural Heuristics for VRP

Most neural heuristics, typically based on GNNs or Transformers, capture the structure of routing problems by treating the coordinates of problem instances as node features (Figure 1, (a)). However, encoding edge features rather than relying solely on node features more closely aligns with practical applications. Early works incorporating edge features include variants of Graph Attention Networks (GAT) [Veličković *et al.*, 2017]. For example, [Chen and Chen, 2021] introduced Edge-featured Graph Attention Networks (EGAT), which consider edge features during message-passing, while [Shi *et al.*, 2020] integrated edge features into attention-based GNNs using "Graph Transformers" for semi-supervised classification. In addition, [Jin *et al.*, 2023a] proposed EdgeFormers, an architecture that processes text edge networks to enhance GNNs for better utilization of edge (text) features.

In the context of routing problems, several edge-based approaches have been explored, such as MatNet [Kwon *et al.*, 2021] and GREAT [Lischka *et al.*, 2024]. Specifically, [Kwon *et al.*, 2021] proposed a Matrix Encoding Network (MatNet) that accepts an encoded distance matrix to solve complex asymmetric traveling salesman (ATSP) and flexible flow shop (FFSP) problems. Meanwhile, [Lischka *et al.*, 2024] introduced the Graph Edge Attention Network (GREAT), an edge-based neural model related to GNNs, which uses highly sparse

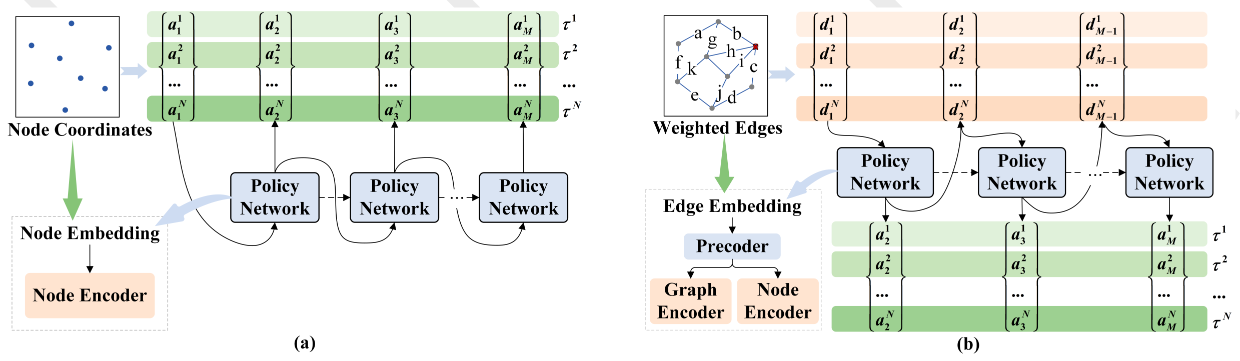

Figure 1: Comparison of node-based and edge-based policy networks, where multiple trajectory $\{\tau^1, \tau^2, \ldots, \tau^N\}$ is pursued in both for inferring solutions. (a) Overall scheme of the node-based methods such as POMO. (b) Overall scheme of the EFormer, which takes edge weights as the input.

graphs to achieve high-quality solutions. Following this line of research, our work also focuses on edge-based methods and proposes a neural network model architecture that improves the solution quality for solving VRPs (Figure 1, (b)).

3 Methodology: EFormer

We introduce the EFormer, a model for solving VRPs that comprises four main components: a *precoder*, a *graph encoder*, a *node encoder*, and a *decoder*. Given a set of edge information, EFormer first applies the precoder to generate temporary node embeddings. To mitigate the heavy computation arising from dense graphs, we adopt a k-nearest neighbor (k-nn) heuristic to sparsify them. It then employs parallel graph and node encoders to process graph and node embeddings in distinct feature spaces, respectively. Finally, the decoder constructs a path by integrating parallel context embeddings and a multi-query mechanism. Figure 2 shows the overall framework of EFormer, while a more detailed version is provided in Appendix A.1.

3.1 Precoder

The precoder processes externally provided edge information to generate node embeddings, effectively serving as a pre-encoding step in EFormer. It comprises a multi-head mixed-score attention layer and a feed-forward (FF) layer, as shown in Figure 2. We introduce its key steps below.

The input's initial representations. Along with the edge weights, we also incorporate a zero vector, a one-hot vector, and the edge weight matrix. This approach is crucial because the one-hot vector is randomly selected from a predefined matrix pool, ensuring that each iteration generates unique embeddings. Consequently, we can supply the same problem instances along with the zero embedding multiple times, enabling flexible instance augmentation.

We generally follow the graph attention networks (GATs) framework [Veličković *et al.*, 2017], which is described in detail in Appendix A.2. Nevertheless, unlike the classic GATs, the attention score for a pair of nodes (i, j) in our method depends not only on $\hat{h}_i$ and $\hat{h}_j$, but also on the edge weight $e(i, j)$. The precoder's update function is defined as:

$$\hat{h}'_i = \mathcal{F}_R\Big(\hat{h}_i, \hat{h}_j, e(i,j) \mid i \in R, j \in M\Big), \quad (1)$$

where the learnable update function $\mathcal{F}$ leverages multi-head attention (MHA). Its aggregation process uses attention scores for each node pair (i, j), which are determined by $\hat{h}_i$ and $\hat{h}_j$. Here, R denotes the set of adjacent nodes of i, and M denotes the set of adjacent nodes of j.

Mixed-score attention. For inputs containing only edge information, it is crucial to incorporate edge weights $e(i, j) \equiv D_{ij}$ in the attention mechanism. Drawing inspiration from [Kwon *et al.*, 2021], we adopt a "Multi-Head Mixed-Score Attention" block to process the edge weight matrix. This block closely follows the MHA module in Transformer, except that the scaled dot-product attention in each head is replaced with mixed-score attention. Specifically, the block integrates the externally provided edge distance matrix D_{ij} with the internally generated attention scores. A small Multilayer Perceptron (MLP) with two inputs and one output determines the optimal way to combine these scores. The resultant mixed scores then pass through a "softmax" stage, thereby retaining matrix-based representation crucial to attention mechanisms.

By performing mixed-score attention on the given edge information, it produces an encoded relationship matrix $h_i^{(P)}$, which acts as the node embeddings for both the graph and node encoders. This process is given by:

$$\hat{h_i}^{(P)} = h_r + \text{mixed-scoreMHA}(h_r, h_c, D_{ij}), \quad (2)$$

$$h_i^{(P)} = \hat{h_i}^{(P)} + \text{FF}(\hat{h_i}^{(P)}), \quad (3)$$

where D_{ij} is edge weight matrix, h_r is zero-vector embedding, and h_c is one-hot vector embedding.

Instance augmentation. Since we initialize each run with a random sequence of one-hot vectors, the weight matrix D_{ij} is encoded differently each time. As a result, the precoder can yield diverse representations of the same problem instance and generate distinct solutions. By providing a new one-hot vector sequence for each run, a multitude of different solutions can be readily obtained simply by repeatedly executing the model. Moreover, this random one-hot embedding strategy naturally aligns with the "instance augmentation" technique proposed in POMO [Kwon *et al.*, 2020]. However, whereas POMO provides ×8 instance augmentation, EFormer can achieve ×N

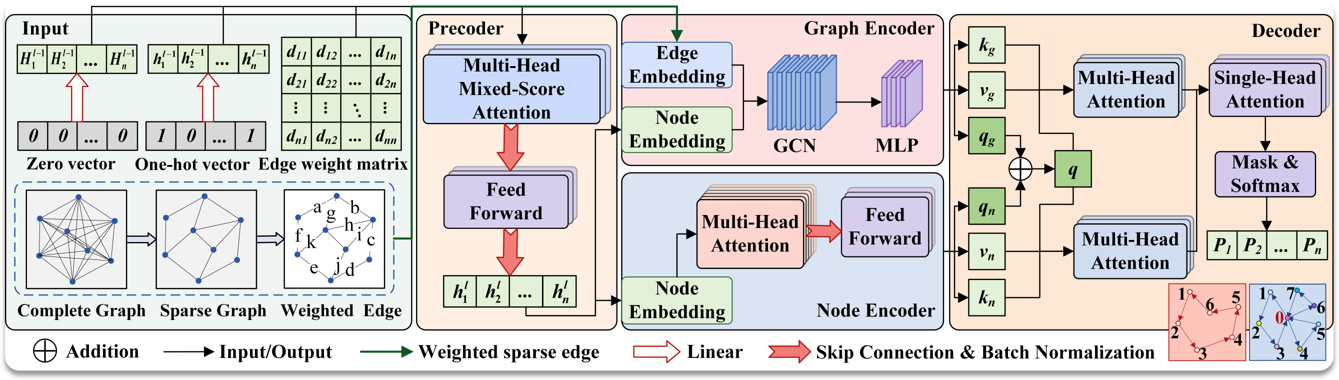

Figure 2: Overview of the EFormer framework. From the input of sparse weighted edges, the *precoder* performs preprocessing, the *graph encoder* and *node encoder* operate in parallel, and finally the *decoder* constructs the complete path.

augmentation. The trade-off is that while this approach yields higher-quality solutions, it also increases runtime.

3.2 Graph Encoder

The graph encoder is a pivotal component of our parallel strategy, comprising residual gated GCN layers and MLP layers. It processes a sparse graph formed by the edge embeddings and node embeddings produced by the precoder.

Graph sparsification. To mitigate the computational burden arising from dense graphs, we employ a k-nn heuristic, in which each node is connected to its k nearest neighbors. The choice of k and its impact on training efficiency are evaluated in the experimental section, where we demonstrate the importance of sparsity for improving training efficiency.

Input layer. The precoder's output $h_i^{(P)}$, serves as the graph encoder's node embedding. We project $h_i^{(P)}$ into h-dimensional feature as $x_i^{l=0} = A_1 h_i^{(P)}$ with $A_1 \in R^h$.

We define an edge adjacency matrix function δ_{ij}^{KNN}, which has a value of 1 if nodes i and j are k-nearest neighbors, a value of 2 if they are self-connected, and a value of 0 otherwise. The edge adjacency matrix δ_{ij}^{KNN} and the edge weight matrix D_{ij} are embedded as $\frac{h}{2}$-dimensional feature vectors. Then we concatenate the two together to get the edge input feature as $e_{ij}^{l=0} = A_2 D_{ij} + b_3 || A_3 \cdot \delta_{ij}^{\text{KNN}}$, where $A_2 \in R^{\frac{h}{2}}$, $A_3 \in R^{\frac{h}{2}}$, b_3 is the bias, and $\cdot||\cdot$ is the concatenation operator.

Residual gated Graph Convolution layer. Let x_i^l and e_{ij}^l represent the node and edge embedding associated with the node i and edge (i,j) at layer l, respectively. In our method, we employ the GCN architecture introduced in [Bresson and Laurent, 2017], to produce the node embedding x_i^{l+1} and edge embedding e_{ij}^{l+1} at the next layer as follows:

$$x_i^{l+1} = x_i^l + \text{ReLu}(\text{BN}(W_1^l x_i^l + \eta_{ij}^l \odot W_2^l x_j^l)), \tag{4}$$

$$e_{ij}^{l+1} = e_{ij}^l + \text{ReLu}(\text{BN}(W_3^l e_{ij}^l + W_4^l x_i^l + W_5^l x_j^l)), \tag{5}$$

$$\eta_{ij}^l = \sum_{j\sim i} \frac{\sigma(e_{ij}^l)}{\sum_{j'\sim i} \sigma(e_{ij'}^l) + \xi}, \tag{6}$$

where $W_*^l \in R^h$, σ is the sigmoid function, ξ is a small value, ReLU is the rectified linear unit, and BN stands for batch normalization [Ioffe, 2015]. Meanwhile, GCN exploits the additive edge feature representation and dense attention map η_{ij}, to make the diffusion process anisotropic on the graph.

MLP layer. In the final layer, MLP modules process both the node and edge embeddings derived from the GCN layers. Here, we specifically focus on the node embedding x_i^l, which inherently incorporates the relevant edge and node information. The MLP outputs h_{ij}^G in $[0,1]^2$ defined as $h_{ij}^G = \text{MLP}(x_i^l)$, where the number of MLP layers is denoted by l_{mlp}.

3.3 Node Encoder

Node encoder processes the node embeddings generated by precoder, using attention mechanisms to embed them into the distinct feature space and produce encoded representations.

Input layer. Unlike the traditional approach [Kool *et al.*, 2019] that relies on node coordinates, the node encoder uses the temporary node embedding $h_i^{(P)}$ derived from the precoder as the input to its attention layer. Let $H^{(0)}$ denote the input to the first layer, then it will be given as $H^{(0)} = h_i^{(P)}$.

Attention layer. Node encoder comprises n attention layers. Each attention layer consists of two sublayers: a MHA sublayer and a FF sublayer [Vaswani, 2017]. Each sublayer incorporates a residual connection [He *et al.*, 2016] and batch normalization (BN). We denote the embedding obtained from each layer as $h_i^{(l)}$, and let $H^{(l-1)} = \left(h_1^{(l-1)}, h_2^{(l-1)}, \ldots, h_n^{(l-1)}\right)$ be the input to the l-th attention layer when $l = \{1, ..., L\}$. The output of the attention layer for the embedding of the i-th node is calculated as follows:

$$\hat{h}_i^{(l)} = \text{BN}\left(h_i^{(l-1)} + \text{MHA}\left(h_i^{(l-1)}, H^{(l-1)}\right)\right), \tag{7}$$

$$h_i^{(l)} = \text{BN}\left(\hat{h}_i^{(l)} + \text{FF}\left(\hat{h}_i^{(l)}\right)\right), \tag{8}$$

where the FF sublayer includes one hidden sublayer and ReLU activation. The above process and the final output can be summarized as follows:

$$H^{(l)} = \text{AttentionLayer}(H^{(l-1)}), \quad h_L^N = H^{(L)}. \tag{9}$$

Note. we employ two parallel encoding components—the graph encoder and node encoder—both of which take node embeddings as (one of the) inputs. One might wonder why the node encoder is needed if the graph encoder can already handle node embeddings. Although using multiple GCN layers can yield powerful encoding representations, it often increases model complexity. By contrast, the node encoder uses fewer layers and leverages attention mechanisms, allowing the model to remain lightweight while still delivering strong performance. Moreover, the attention mechanisms have been shown to be highly effective for encoding node embeddings. Empirically, this parallel architecture outperforms approaches that rely solely on GCNs or a single attention-based encoder.

3.4 Decoder

During the decoding phase, the edge embeddings and node embeddings produced by the graph encoder and node encoder, respectively, are processed in parallel. We use superscripts to differentiate the sources of information: G denotes edge embeddings, while N denotes node embeddings. Initially, two separate sets of keys and values are extracted from the edge and node embeddings. In parallel, the current contextual embedding is derived from the current state in combination with these two sets of embeddings.

The embeddings of all the starting nodes h^G_{first}, h^N_{first} (i.e., the nodes selected in the first step) and the embeddings of the target nodes h^G_{last}, h^N_{last} (i.e., the currently selected nodes) are concatenated to form two sets of temporary queries (i.e., q^G, q^N). Afterwards, the two sets of context embeddings are merged to create the final query as follows:

$$q^G = W^G_1 h^G_{first} + W^G_2 h^G_{last}, \tag{10}$$

$$q^N = W^N_1 h^N_{first} + W^N_2 h^N_{last}, \tag{11}$$

$$q = q^G + q^N, \tag{12}$$

where the subscript $first$ denotes the fixed start node, and the subscript $last$ denotes the current target node. W^G_1, W^G_2, W^N_1 and W^N_2 are learnable matrices used to recast the start node embeddings h^G_{first}, h^N_{first} and the current target node embeddings h^G_{last}, h^N_{last}, respectively.

Next, we apply the MHA mechanism to each set of context embeddings separately, producing two outputs, A^G and A^N:

$$A^G = \mathrm{MHA}(q, k^G, v^G), \tag{13}$$

$$A^N = \mathrm{MHA}(q, k^N, v^N), \tag{14}$$

We then apply two linear layers W^G_3 and W^N_3 to map A^G and A^N, respectively. Subsequently, we compute a score via two sets of single-head attention layers, apply the tanh function to clip the score, and mask any visited nodes. The resulting score u_j for node j is given by:

$$u_j = \begin{cases} C \cdot \tanh\left(\frac{(W^G_3 A^G + W^N_3 A^N)(k^G_j + k^N_j)}{\sqrt{d_k}}\right), \text{if } j \text{ unvisited} \\ -\infty, \text{otherwise} \end{cases} \tag{15}$$

where W^G_3 and W^N_3 are learnable matrices, u_j is the score for node j, and d_k is determined by the embedding dimension. We then apply the softmax function to calculate the probability p_j of selecting node j. At each decoding step j, the next node is selected based on its probability p_j. Repeating this process n times yields the complete solution $\tau = \{\tau^1, \cdots, \tau^n\}^T$:

$$p_j = \mathrm{softmax}(u_j). \tag{16}$$

3.5 Training

Since EFormer can be easily integrated into a variety of autoregressive solvers, we adopt the same reinforcement learning training method as POMO. We train the EFormer model using the REINFORCE algorithm [Williams, 1992]. We sample a set of n trajectories $\{\tau^1, \cdots, \tau^n\}$, calculate the reward $f(\tau^i)$ for each, and employ approximate gradient ascent to maximize the expected return $\mathcal{L}$. The gradient of the total training loss $\mathcal{L}$ can be approximated as follows:

$$\nabla_\theta \mathcal{L}(\theta) \approx \frac{1}{n}\sum_{i=1}^{n}[(f(\tau^i) - b^i(s))\nabla \log p_\theta(\tau^i|s)], \tag{17}$$

where $b^i(s)$ is commonly set as the average reward of those m trajectories, serving as a shared baseline:

$$b^i(s) = b_{\text{shared}}(s) = \frac{1}{n}\sum_{i=1}^{n} f(\tau^i), \text{ for all } i \tag{18}$$

$$\text{where } p_\theta(\tau^i|s) = \prod_{t=2}^{M} p_\theta(a^i_t|s, a^i_{1:t-1}). \tag{19}$$

4 Experiment

We empirically evaluate our proposed EFormer model on TSP and CVRP of various sizes and distributions, comparing it against both learning-based and classical solvers. Our code is publicly available[2].

Basic Settings. We compare EFormer with: **(1) Classical solvers:** Concorde [Cook *et al.*, 2011], LKH3 [Helsgaun, 2017], HGS [Vidal, 2022], and OR-Tools [Perron and Furnon, 2023]; **(2) Heatmap-based method:** GCN-BS [Joshi *et al.*, 2019]; **(3) Edge-based neural heuristics:** MatNet [Kwon *et al.*, 2021] and GREAT [Lischka *et al.*, 2024]. We follow the standard data generation procedures from prior work [Kool *et al.*, 2019] to create training and testing datasets for TSP and CVRP, where distances between nodes are calculated and provided as inputs accordingly. Each training epoch samples 100,000 random instances, while a separate set of 10,000 uniformly generated instances is used for testing. Optimal solutions for TSP are obtained via the Concorde solver, and those for CVRP using LKH3. We adopt the POMO inference algorithm [Kwon *et al.*, 2020] and report both the optimality gap and inference time. For EFormer specifically, we present results for greedy inference (×1) and instance augmentation (×8 and ×128). Details of the experimental setup and additional baseline information can be found in Appendix B.1.

Table 1 presents our main results on uniformly distributed TSP and CVRP instances. For TSP, our proposed EFormer achieves excellent greedy inference (x1) performance across

[2]https://github.com/Regina921/EFormer

	TSP20			TSP50			TSP100		
Method	Len.	Gap(%)	Time(m)	Len.	Gap(%)	Time(m)	Len.	Gap(%)	Time(m)
Concorde	3.831	0.000	4.43	5.692	0.000	23.53	7.763	0.000	66.45
LKH3	3.831	0.000	2.78	5.692	0.000	17.21	7.763	0.000	49.56
OR-Tools	3.864	0.864	1.16	5.851	2.795	10.75	8.057	3.782	39.05
GCN-Greedy	3.948	3.078	0.32	5.968	4.856	1.34	8.537	9.966	4.09
GCN-BS	3.862	0.825	0.81	5.732	0.712	4.33	8.170	5.234	4.31
GCN-BS*	3.831	0.004	21.41	5.700	0.141	35.46	7.955	2.477	63.05
MatNet(×1)	3.832	0.044	0.11	5.709	0.303	0.13	7.836	0.940	0.52
MatNet(×8)	3.831	0.002	0.22	5.694	0.050	1.24	7.795	0.410	5.28
MatNet(×128)	3.831	0.000	5.71	5.692	0.013	16.47	7.776	0.170	60.11
GREAT(×1)#	-	-	-	-	-	-	7.850	1.210	2.00
GREAT(×8)#	-	-	-	-	-	-	7.820	0.810	18.00
EFormer(×1)	3.831	0.018	0.04	5.699	0.130	0.26	7.788	0.324	1.22
EFormer(×8)	3.831	0.000	0.15	5.692	0.011	1.34	7.772	0.115	6.81
EFormer(×128)	3.831	**0.000**	4.72	5.692	**0.001**	25.81	7.767	**0.045**	66.55

	CVRP20			CVRP50			CVRP100		
Method	Len.	Gap(%)	Time(m)	Len.	Gap(%)	Time(m)	Len.	Gap(%)	Time(m)
LKH3	6.117	0.000	2.15h	10.347	0.000	8.52h	15.647	0.000	13.46h
HGS	6.112	-0.079	1.48h	10.347	-0.001	4.67h	15.584	-0.401	6.54h
OR-Tools	6.414	4.863	2.37	11.219	8.430	19.35	17.172	9.749	2.61h
GCN-Greedy	6.471	5.794	0.27	11.130	7.567	2.05	16.948	8.314	5.24
GCN-BS	6.284	2.740	0.26	10.786	4.248	2.11	16.487	5.371	5.45
GCN-BS*	6.192	1.232	20.78	10.636	2.796	38.36	16.243	3.811	78.78
MatNet(×1)	6.172	0.907	0.11	10.787	4.253	0.21	16.280	4.401	1.02
MatNet(×8)	6.146	0.469	0.58	10.635	2.787	1.23	16.117	3.356	4.70
MatNet(×128)	6.131	0.229	9.93	10.538	1.847	17.93	15.989	2.530	66.05
EFormer(×1)	6.147	0.490	0.04	10.457	1.067	0.25	15.844	1.259	0.98
EFormer(×8)	6.123	0.098	0.28	10.414	0.650	1.69	15.776	0.830	6.86
EFormer(×128)	6.116	**-0.017**	14.84	10.393	**0.447**	24.47	15.735	**0.563**	85.65

Table 1: Experimental results on TSP and CVRP with uniformly distributed instances. The results of methods with an asterisk (#) are directly obtained from the original paper. BS: Beam search, BS*: Beam search and shortest tour heuristic

	TSP50		CVRP50			TSP50		CVRP50	
Method	Len.	Gap(%)	Len.	Gap(%)	Method	Len.	Gap(%)	Len.	Gap(%)
OPT	5.692	0.000	10.347	0.000	OPT	5.692	0.000	10.347	0.000
K=10 (×1)	5.698	**0.117**	10.476	1.250	w.o. precoder(×1)	5.702	0.181	10.642	2.854
K=20 (×1)	5.699	0.130	10.474	**1.231**	w.o. node encoder(×1)	5.702	0.185	10.517	1.640
K=30 (×1)	5.699	0.135	10.484	1.321	w.o. graph encoder(×1)	5.705	0.233	10.499	1.466
K=40 (×1)	5.700	0.147	10.486	1.346	w.o. gcn(×1)	5.707	0.276	10.502	1.501
K=50 (×1)	5.699	0.133	10.485	1.332	**EFormer(×1)**	5.699	**0.130**	10.474	**1.231**
K=10 (×8)	5.692	0.012	10.424	0.746	w.o. precoder(×8)	-	-	-	-
K=20 (×8)	5.692	**0.011**	10.422	**0.725**	w.o. node encoder(×8)	5.693	0.026	10.442	0.918
K=30 (×8)	5.692	0.012	10.423	0.734	w.o. graph encoder(×8)	5.693	0.020	10.431	0.817
K=40 (×8)	5.693	0.020	10.423	0.740	w.o. gcn(×8)	5.693	0.021	10.428	0.782
K=50 (×8)	5.692	0.017	10.422	0.731	**EFormer(×8)**	5.692	**0.011**	10.422	**0.725**

Table 2: Ablations of various K values and four key elements of EFormer on uniformly distributed instances.

various instance sizes, all within a relatively reasonable inference time. Additionally, we perform inferences with instance augmentation (×8, ×128), which significantly outperform other neural heuristics. Notably, our x8 augmentation even surpasses MatNet's x128 augmentation. For TSP100, the optimality gap can be as low as 0.0453% when using x128 augmentation. EFormer thus clearly outperforms other edge-based methods, whether using greedy inference or instance augmentation.

For CVRP, we extend and refine the two learning-based baselines from their original formulations while retaining their model structure and parameters to ensure effective CVRP solutions. Regardless of whether greedy inference (×1) or instance augmentation (×8 or ×128) is employed, our EFormer consistently outperforms other edge-based methods. With ×128 augmentation, EFormer achieves an optimality gap of 0.5633% on CVRP100. Thus, EFormer demonstrates superior performance over competing approaches under both greedy inference and augmented settings. Given a larger time budget, EFormer can leverage additional instance augmentation to further enhance performance across all instances.

	TSPLIB1-100		TSPLIB101-300		TSPLIB301-500		CVRPLIB1-100		CVRPLIB101-300		CVRPLIB301-500	
Method	Len.	Gap(%)	Len.	Gap(%)	Len.	Gap(%)	Len.	Gap(%)	Len.	Gap(%)	Len.	Gap(%)
OPT	19499.583	0.000	40129.375	0.000	43694.666	0.000	925.800	0.000	33184.483	0.000	82903.857	0.000
GCN-Greedy	25371.528	30.242	63125.529	55.716	81143.342	83.080	1233.255	34.732	44684.202	45.069	138044.859	63.597
GCN-BS	23906.973	22.631	58213.359	46.951	76546.164	72.946	1167.058	27.909	42773.044	38.466	120532.679	49.664
GCN-BS*	21195.267	10.824	55853.973	39.104	72865.950	64.648	1087.529	18.369	41403.524	34.012	114962.535	42.366
MatNet(×1)	20076.205	3.880	46876.768	17.868	63544.470	46.192	1098.828	21.017	38946.522	20.351	103278.318	24.309
MatNet(×8)	19761.313	1.786	45638.627	14.479	62427.744	42.206	1072.623	14.901	38218.037	17.657	98912.043	19.460
EFormer(×1)	19741.575	1.698	45396.807	13.599	61415.142	36.562	1107.504	18.766	38477.709	12.485	98010.813	21.023
EFormer(×8)	19662.845	**0.837**	43360.592	**8.805**	56111.793	**26.924**	1019.698	**9.885**	36968.746	**9.784**	94612.575	**15.325**

Table 3: Experimental results on TSPLib and CVRPLib.

4.1 Ablation Study

We conduct two sets of ablation studies to clarify essential design choices in our method. Specifically, we focus on: (1) the selection of the hyperparameter $K = 20$ for the k-nn graph sparsification, and (2) the necessity of using three encoders.

K value selection. In the classic knn-based sparsification approach, the hyperparameter K determines how many edges are retained for each node ($K \times N$, where N is the TSP size). We compare various K values (10, 20, 30, 40, and 50) on TSP50 and CVRP50; the results are shown in Table 2. For TSP50, the optimality gaps for K = 10, 20, and 30 are similar, with K = 20 performing slightly better. For CVRP50, K = 20 produces the highest solution quality. Overall, K = 20 appears to capture nearly all the best solutions for TSP50 and CVRP50, so we set K = 20 in all subsequent experiments.

The necessity of three encoders. Table 2 shows the ablation results for EFormer and four variants. The first variant removes the precoder, retaining only the node encoder and graph encoder (denoted by w.o. precoder in the table). Without precoder, the model cannot re-encode the same problem instance multiple times; hence, no additional instance augmentation is performed. The second variant removes the node encoder, keeping only the precoder and graph encoder (denoted by w.o. node encoder). The third variant removes the graph encoder, retaining the precoder and node encoder (denoted by w.o. graph encoder). The fourth variant removes the GCN module from the graph encoder but retains the precoder, node encoder, and MLP module of graph encoder (denoted by w.o. gcn). Comparing the third and fourth variants highlights the effectiveness of our parallel dual-encoder structure. As shown in Table 2, EFormer outperforms all variants, indicating that each component contributes positively to the model.

4.2 Generalization

We assess the generality of our proposed EFormer from three perspectives: 1) real-world TSPLIB and CVRPLIB benchmarks, 2) larger-scale instances, and 3) different distributions.

Generalization to TSPLIB and CVRPLIB. Table 3 summarizes the results on real-world TSPLIB [Reinelt, 1991] and CVRPLIB [Uchoa *et al.*, 2017] instances of various sizes and distributions. EFormer performs best on instances with up to 100 nodes and ranks second for 101–300 nodes, demonstrating excellent generalization on both TSPLIB and CVRPLIB. It also outperforms GCN-BS and MatNet on all instances.

Generalization to larger-scale instances. Appendix B.2 shows the performance of EFormer on TSP and CVRP instances with up to 500 nodes. Despite device limitations, EFormer significantly surpasses other edge-based methods, highlighting its robust generalization even when only edge information is available.

Generalization across different distributions. We further evaluate EFormer on TSP and CVRP instances from explosion, grid, and implosion distributions. Appendix B.3 presents the results, indicating that EFormer consistently outperforms GCN-BS and MatNet across all three distributions. Moreover, it maintains strong performance not only on the uniform distribution but also under varying distribution scenarios, underscoring its solid generalization capabilities.

4.3 Node-based EFormer

Our EFormer architecture is highly flexible, enabling it to address VRPs using node coordinates as the only inputs. To compare the performance of EFormer-based solvers with other established methods, we introduce a variant called EFormer-node, which is tested on the traditional node-coordinate setting. Experimental results indicate that EFormer-node delivers competitive performance compared to other established neural heuristics. For further details on the model architecture and experimental findings, please refer to Appendix C. Additionally, we also solve ATSP based on our EFormer framework, and the detailed experimental results are presented in Appendix D.

5 Conclusion

In this paper, we introduce a novel Edge-based Transformer (EFormer) model designed to solve VRPs in an autoregressive way that utilizes edge information as input. Our integrated architecture employs three encoders that work in concert to efficiently capture and process edge information. By adopting a parallel encoding approach, we encode different types of information in separate feature spaces, thereby enhancing the global strategy. Extensive experiments on both uniformly generated synthetic instances and real-world benchmarks demonstrate EFormer's strong performance. Compared to node-based approaches, edge-based methods exhibit greater flexibility and applicability to real-world scenarios. Looking ahead, we plan to investigate lightweight architectural designs for EFormer to improve its scalability across a broader range of problem sizes. Another promising direction is to develop a unified learning-based framework that can operate effectively on both edge and node, where both are available as inputs.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62372081, the Young Elite Scientists Sponsorship Program by CAST under Grant 2022QNRC001, the Liaoning Provincial Natural Science Foundation Program under Grant 2024010785-JH3/107, the Dalian Science and Technology Innovation Fund under Grant 2024JJ12GX020, the Dalian Major Projects of Basic Research under Grant 2023JJ11CG002 and the 111 Project under Grant D23006. This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031).

References

[Ausiello *et al.*, 2012] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

[Barrett *et al.*, 2020] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on AI*, volume 34, pages 3243–3250, 2020.

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

[Brophy and Voigt, 2014] Jennifer AN Brophy and Christopher A Voigt. Principles of genetic circuit design. *Nature methods*, 11(5):508–520, 2014.

[Chen and Chen, 2021] Jun Chen and Haopeng Chen. Edge-featured graph attention network. *arXiv preprint arXiv:2101.07671*, 2021.

[Cheng *et al.*, 2023] Hanni Cheng, Haosi Zheng, Ya Cong, Weihao Jiang, and Shiliang Pu. Select and optimize: Learning to solve large-scale tsp instances. In *International Conference on Artificial Intelligence and Statistics*, pages 1219–1231. PMLR, 2023.

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

[Drakulic *et al.*, 2023] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for generalizable neural combinatorial optimization. *arXiv preprint arXiv:2301.03313*, 2023.

[Elgarej *et al.*, 2021] Mouhcine Elgarej, Mansouri Khalifa, and Mohamed Youssfi. Optimized path planning for electric vehicle routing and charging station navigation systems. In *Research Anthology on Architectures, Frameworks, and Integration Strategies for Distributed and Cloud Computing*, pages 1945–1967. IGI Global, 2021.

[Fu *et al.*, 2021] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on AI*, volume 35, pages 7474–7482, 2021.

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

[Helsgaun, 2017] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

[Huang *et al.*, 2025] Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. In *International Conference on Learning Representations*, 2025.

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

[Ioffe, 2015] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[Jin *et al.*, 2023a] Bowen Jin, Yu Zhang, Yu Meng, and Jiawei Han. Edgeformers: Graph-empowered transformers for representation learning on textual-edge networks. *arXiv preprint arXiv:2302.11050*, 2023.

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

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

[Khalil *et al.*, 2017] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *In NeurIPS*, 30, 2017.

[Kim *et al.*, 2021] Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. *In NeurIPS*, 34:10418–10430, 2021.

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

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Konstantakopoulos *et al.*, 2022] Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational research*, 22(3):2033–2062, 2022.

[Kool *et al.*, 2019] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2019.

[Kool *et al.*, 2022] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. In *International conference on integration of constraint programming, artificial intelligence, and operations research*, pages 190–213. Springer, 2022.

[Kwon *et al.*, 2020] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *In NeurIPS*, 33:21188–21198, 2020.

[Kwon *et al.*, 2021] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *In NeurIPS*, 34:5138–5149, 2021.

[Li *et al.*, 2018] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *In NeurIPS*, 31, 2018.

[Li *et al.*, 2021] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *In NeurIPS*, 34:26198–26211, 2021.

[Lin *et al.*, 2024] Zhuoyi Lin, Yaoxin Wu, Bangjian Zhou, Zhiguang Cao, Wen Song, Yingqian Zhang, and Senthilnath Jayavelu. Cross-problem learning for solving vehicle routing problems. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 6958–6966, 2024.

[Lischka *et al.*, 2024] Attila Lischka, Jiaming Wu, Morteza Haghir Chehreghani, and Balázs Kulcsár. A great architecture for edge-based graph problems like tsp. *arXiv preprint arXiv:2408.16717*, 2024.

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

[Luo *et al.*, 2023] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *In NeurIPS*, 36:8845–8864, 2023.

[Ma *et al.*, 2021] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *In NeurIPS*, 34:11096–11107, 2021.

[Min *et al.*, 2024] Yimeng Min, Yiwei Bai, and Carla P Gomes. Unsupervised learning for solving the travelling salesman problem. *In NeurIPS*, 36, 2024.

[Nazari *et al.*, 2018] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *In NeurIPS*, 31, 2018.

[Nowak *et al.*, 2017] Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. A note on learning algorithms for quadratic assignment with graph neural networks. *stat*, 1050:22, 2017.

[Perron and Furnon, 2023] Laurent Perron and Vincent Furnon. Or-tools. https://developers.google.com/optimization/routing, 2023. Accessed: 2024-08.

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

[Scarselli *et al.*, 2008] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[Shi *et al.*, 2020] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[Uchoa *et al.*, 2017] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[Vaswani, 2017] A Vaswani. Attention is all you need. *In NeurIPS*, 2017.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Vidal, 2022] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[Wang *et al.*, 2021] Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. *In NeurIPS*, 34:21453–21466, 2021.

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

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

[Xin *et al.*, 2021] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *In NeurIPS*, 34:7472–7483, 2021.

[Zhang *et al.*, 2025] Ni Zhang, Jingfeng Yang, Zhiguang Cao, and Xu Chi. Adversarial generative flow network for solving vehicle routing problems. In *International Conference on Learning Representations*, 2025.