Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems | School of Computing and Information Systems

1-2023

Learning feature embedding refiner for solving vehicle routing problems

Jingwen LI

Yining MA

Zhiguang CAO
Singapore Management University, zgcao@smu.edu.sg

Yaoxin WU

Wen SONG

See next page for additional authors

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the OS and Networks Commons

Citation

LI, Jingwen; MA, Yining; CAO, Zhiguang; WU, Yaoxin; SONG, Wen; ZHANG, Jie; and CHEE, Yeow Meng. Learning feature embedding refiner for solving vehicle routing problems. (2023). *IEEE Transactions on Neural Networks and Learning Systems*. 1-13.
Available at: https://ink.library.smu.edu.sg/sis_research/8087

This Journal Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Author

Jingwen LI, Yining MA, Zhiguang CAO, Yaoxin WU, Wen SONG, Jie ZHANG, and Yeow Meng CHEE

This journal article is available at Institutional Knowledge at Singapore Management University: https://ink.library.smu.edu.sg/sis_research/8087

ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/371034010

Learning Feature Embedding Refiner for Solving Vehicle Routing Problems

Article *in* IEEE Transactions on Neural Networks and Learning Systems · May 2023

DOI: 10.1109/TNNLS.2023.3285077

CITATIONS
0

READS
262

7 authors, including:

Li Jingwen
NUS
9 PUBLICATIONS **147** CITATIONS
SEE PROFILE

Yining Ma
National University of Singapore
16 PUBLICATIONS **237** CITATIONS
SEE PROFILE

Zhiguang Cao
Singapore Management University
78 PUBLICATIONS **2,252** CITATIONS
SEE PROFILE

Wen Song
Shandong University
63 PUBLICATIONS **1,136** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Coding for flash memories View project

Framework for Rapid Simulation of Complex Business Systems View project

All content following this page was uploaded by Li Jingwen on 09 June 2023.

The user has requested enhancement of the downloaded file.

Learning Feature Embedding Refiner for Solving Vehicle Routing Problems

Jingwen Li, Yining Ma, Zhiguang Cao, Yaoxin Wu, Wen Song, Jie Zhang, Yeow Meng Chee

***Abstract*—While the encoder-decoder structure is widely used in the recent neural construction methods for learning to solve vehicle routing problems, they are less effective in searching solutions due to deterministic feature embeddings and deterministic probability distributions. In this paper, we propose the Feature Embedding Refiner (FER) with a novel and generic encoder-refiner-decoder structure to boost the existing encoder-decoder structured deep models. It is *model-agnostic* that the encoder and the decoder can be from any pre-trained neural construction method. Regarding the introduced *refiner* network, we design its architecture by combining standard GRU cell with two new layers, i.e., an accumulated graph attention (AGA) layer and a gated nonlinear (GNL) layer. The former extracts dynamic graph topological information of historical solutions stored in a diversified solution pool to generate aggregated pool embeddings that are further improved by the GRU, and the latter adaptively refines the feature embeddings from the encoder with the guidance of the improved pool embeddings. To this end, our FER allows current neural construction methods to not only iteratively refine the feature embeddings for boarder search range but also dynamically update the probability distributions for more diverse search. We apply FER to two prevailing neural construction methods including AM and POMO to solve the travelling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP). Experimental results show that our method achieves lower gaps and better generalization than the original ones, and also exhibits competitive performance to the state-of-the-art neural improvement methods.**

***Index Terms*—Vehicle routing problems, neural combinatorial optimization, encoder-decoder structure, reinforcement learning**

I. Introduction

DUE to the NP-hard nature, vehicle routing problems (VRPs) including the travelling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP) are intractable to solve optimally through exact solvers [1]. As desirable alternatives, heuristic methods (e.g., [2], [3]) which hinge on certain hand-crafted rules to simplify the search, are always adopted in industry to find near-optimal solutions with much less computational costs. On the other hand, given the remarkable success of deep neural networks in computer vision and natural language processing, it is commonly known that deep models without much human guidance could significantly outperform the hand-crafted ones [4]. Inspired by this superiority, more and more endeavours have been carried out to explore *neural* heuristics for VRPs [5]–[8], which apply feature learning strategy [9] to leverage deep models to automatically learn the rules in heuristics rather than using the hand-crafted ones.

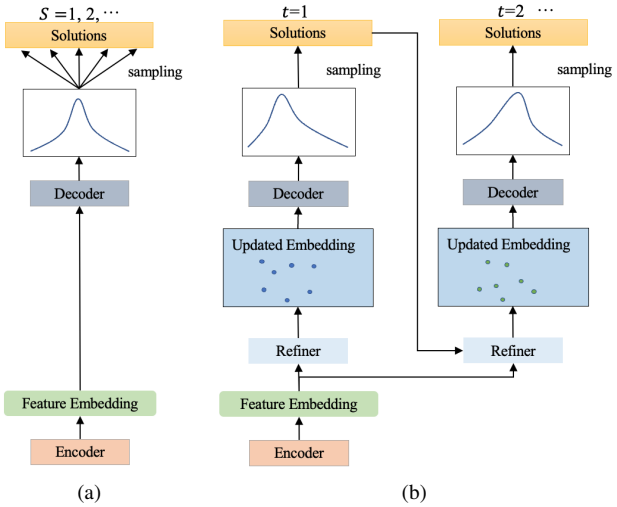


Fig. 1: Structures of neural heuristics. (a) Existing encoder-decoder structure; (b) Our encoder-*refiner*-decoder structure.

In such neural heuristics, the encoder-decoder structure is widely exploited by the deep models to parameterize the probability distribution for sampling a solution. Further trained in the fashion of advanced reinforcement learning or supervised learning, the encoder-decoder structure performs fairly well for solving VRPs, especially in learning neural *construction* heuristics [10], which sequentially decides the next node to visit. Typically, as shown in Fig. 1(a), the encoder learns the representation based on the problem-specific information and produces feature embeddings for all nodes, and the decoder produces a probability distribution over nodes based on the feature embeddings. Given the probability distribution, multiple solutions could be sampled, and the best one will

This work was supported by the National Natural Science Foundation of China under Grant 62102228, the Natural Science Foundation of Shandong Province under Grant ZR2021QF063, and the Agency for Science Technology and Research Career Development Fund under Grant C222812027. *(Jingwen Li and Yining Ma contributed equally to this work.) (Corresponding author: Wen Song.)*

Jingwen Li is with the Department of Computer Science, Sichuan Normal University, Chengdu 610101, China (lijingwen@sicnu.edu.cn).

Yining Ma and Yeow Meng Chee are with the Department of Industrial Systems Engineering and Management, College of Design and Engineering, National University of Singapore, Singapore (yiningma@u.nus.edu, ymchee@nus.edu.sg).

Zhiguang Cao is with the School of Computing and Information Systems, Singapore Management University, Singapore (zhiguangcao@outlook.com).

Yaoxin Wu is with the Faculty of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Eindhoven 5600 MB, The Netherlands (wyxacc@hotmail.com).

Wen Song is with the Institute of Marine Science and Technology, Shandong University, Qingdao 266237, China (wensong@email.sdu.edu.cn).

Jie Zhang is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore (zhangj@ntu.edu.sg).

be retrieved as the final output. However, as the search space may exponentially increase with the problem scales, an effective and diverse exploration is crucial to find high-quality solutions with limited computation time [11], [12]. To this end, the currently used encoder-decoder structure is not optimal for the neural *construction* methods in our views. Specifically, it suffers from two limitations, i.e., deterministic feature embeddings and deterministic probability distribution. Regarding the former, the feature embeddings are fixed during the whole sampling process, which narrows the search range and ignores the impacts of the sampled solutions. Regarding the latter, although multiple solutions are sampled, most of them might be intrinsically identical in light of the unchanged distribution, which may seriously impair the search diversity.

To address the two limitations, we propose the *Feature Embedding Refiner (FER)* with a novel and generic *encoder-refiner-decoder* structure, where the *refiner* is added between the encoder and decoder to better synergize them as depicted in Fig. 1(b). Given a (pre-trained) neural construction model, FER iteratively refines the feature embeddings generated from the encoder and reconstructs a solution(s) with dynamic probability distributions accordingly via the decoder. Specifically, the *refiner* is composed of an *accumulated graph attention* (AGA) layer, a *gated recurrent units* (GRU) layer and a *gated nonlinear* (GNL) layer. At each step, the AGA first extracts the dynamic graph topological features of historical solutions stored in a diversified solution pool to derive aggregated pool embeddings. The pool embeddings are further improved by the GRU to absorb more instructive and mroe global information from previous steps. Afterwards, the GNL learns to adaptively refine the pre-trained feature embeddings (from the construction encoder) using the improved pool embeddings for more exploration. In doing so, our FER is not only able to exploit the historical solutions seen so far to effectively refine the feature embeddings at each step for broader search range, but also dynamically update the probability distribution accordingly for more diverse search. We also note that there is another line of works, i.e., neural *improvement* methods [13]–[15], which also leverage deep models to iteratively improve a complete initial solution through neighbourhood search. However, they usually need to select the operator(s) depending on the problem-specific properties, and only update the solution locally which requires a large amount of iterations. While our FER is more generic, and could reconstruct a complete solution at each step more freely. The experimental results also demonstrate that our FER performs favorably against the state-of-the-art neural improvement methods, especially with limited number of iterations.

Our contributions are summarized as follows: 1) We propose the *Feature Embedding Refiner* (FER) with a novel and generic *encoder-refiner-decoder* structure for boosting existing *encoder-decoder* structured neural construction models for routing problems. It is designed to be *model-agnostic*, so that the encoder and the decoder can be from any pre-trained models; 2) We design the architecture of the newly added *refiner* network by combining standard GRU cell with two new layers, i.e., an AGA layer and a GNL layer. Given the sampled historical solutions in a diversified pool, the former effectively extracts dynamic graph topological information to generate the aggregated pool embeddings that are further improved by GRU cell. The latter adaptively refines the feature embeddings from the encoder with the guidance of the improved pool embeddings for more exploration. In doing so, the *refiner* enables the policy to achieve dynamic feature embeddings for broader search range and further dynamic probability distributions for more diverse search; 3) We evaluate our FER by applying it to two prevailing neural construction models, i.e., AM [16] and POMO [10]. Extensive experimental results on both synthesized and benchmark instances of routing problems (TSP and CVRP) well verify the superiority of our FER to existing encoder-decoder structured construction models as well as other state-of-the-art learning-based baselines.

The remaining of this paper is organized as follows. Section II reviews related works. Section III presents problem formulations. Section IV introduces our FER method in detail. Section V reports the experimental results. Finally, Section VI concludes the paper.

II. Related work

In this section, we briefly review recent works in learning neural *construction* and *improvement* heuristics for routing problems, as well as some other learning based methods.

A. Neural Construction Methods

Starting from an empty solution, neural construction methods learn to iteratively add node to a partial solution to construct a complete one, where deep neural networks are exploited to produce a distribution over the permutation of nodes. Among the seminal works, Vinyals et al. [17] presented the first attempt where Pointer Network was proposed to solve TSP with supervised learning. It was then extended to reinforcement learning [18] and CVRP [14]. Other than RNNs used in the Pointer Network, graph neural networks were also leveraged in [19], [20]. For example in [20], graph conventional network (GCN) was exploited to compute the probability of each edge appearing in the optimal TSP tour. With recent developments of the self-attention mechanism [4], the Attention Model (AM) [16] adopted a Transformer-style network which follows the encoder-decoder structure to learn a construction model. It exhibited favorable performance in solving various routing problems, and was recognized as one of the milestones in this field. Different from the single decoder used in AM, Xin et al. [21] proposed a Multi-Decoder AM (MDAM) to learn multiple decoding policies to improve the solution quality. In another work, POMO (Policy Optimization with Multiple Optima) [10], which is recognized as the current state-of-the-art, improved AM by forcing diverse rollouts and explored data augment techniques. Although consuming short time for inference, these methods usually require post-processing procedures to ensure more desirable solution quality, such as sampling [14], [16], [22], active search [18], beam search [20], [21], or data augmentation [10]. However, they may suffer from limited efficiency and diversity since the feature embeddings and probability distributions are *fixed* during the whole process.

B. Neural Improvement Methods

As another line of research, neural improvement methods learn to search high-quality solutions by iteratively improving a complete initial solution until reaching a step limit. In [23], a neural heuristic was proposed which learns to pick local search operations as well as the local components of the current solution to perform rewriting. A neural large neighbourhood search (NLNS) method was proposed in [24] which learns to repair solutions following the idea of neighbourhood search. Wu et al. [13] proposed a Transformer based improvement heuristic to pick node pairs for a given local operator. It was extended to Dual-Aspect Collaborative Transformer (DACT) [15] to better combine embeddings of the node and positional features. Together with a novel cyclic positional encoding method, DACT has achieved the state-of-the-art performance among existing neural improvement methods. However, it still suffers from long computational time caused by large numbers of iterative steps. Different from the above works that need less domain expertise, Lu et al. [25] introduced the L2I framework based on a number of problem-specific features and operators. Though the solution quality found by L2I outperforms LKH, one of the strongest traditional solver, in solving CVRP, its computation time could be prohibitively longer than other improvement methods (e.g., DACT [15] and Wu et al. [13]).

C. Other Learning-based Methods

The CVAE-opt-DE in [26] leveraged conditional variational autoencoder to learn a latent search space for routing problems, based on which differential evolution (DE) was adopted to search high-quality solutions. A fully convolutional network (FCN) was adopted to solve TSP with up to 12 customers in [27] with the optimal solutions as the labeled data, which is limited to solve large-scale instances and more constrained routing problems, e.g., CVRP. More recently, the LCP (learning collaborative policies) proposed in [28] combined both construction and improvement methods to solve routing problems, which leveraged a *seeder* to construct diverse candidate solutions and a *reviser* to improve each candidate solution. However, their performance is still inferior to POMO [10] in terms of solution quality and inference time.

Different from above neural works that usually take an encoder-decoder structure, in this paper, we propose the FER with a novel encoder-refiner-decoder structure to iteratively improve an encoder-decoder structured construction method. It allows the neural construction methods to not only iteratively refine the feature embeddings based on sampled historical solutions for boarder search range, but also dynamically update the probability distributions for more diverse search.

III. Problem formulation

In this section, we introduce the studied vehicle routing problems, i.e., TSP and CVRP, and formulate the process of applying our FER to solve them as a Markov Decision Process.

Given a set of customer nodes V indexed by $i = 1, 2, ..., N$, TSP aims to optimize a tour to visit each node exactly once with the objective of minimizing the total travel cost (length). With an additional depot node 0 in V, CVRP aims to optimize routes for a fleet of identical vehicles with capacity to serve a set of customers with demands under the constraints that: 1) each customer must be visited exactly once while the depot could be visited multiple times; 2) for each vehicle, the total demands from customers on its route cannot exceed its capacity. Formally, we define the solution to a target routing problem (i.e., TSP or CVRP) as a directed graph $\delta = (V, E)$, where each element $\{i, j\} \in E$ is a directed edge from node i to node j. Let $C = C(\delta) = \sum_{\{i,j\}\in\delta} d_{ij}$ be the objective value (cost) of solution δ, where d_{ij} refers to the Euclidean distance between node i and node j.

Given an encoder-decoder based construction model, our task is to sample it for T steps, and retrieve the best solution found in the process. To better exploit the sampled historical solutions at step t, we maintain a solution pool denoted as $P_t = \{\delta_1^t, ..., \delta_K^t, C_1^t, ..., C_K^t\}$ which includes at most K solutions and their costs. We model the above search process as a Markov Decision Process (MDP) defined as follows.

State. The state s_t defined in Eq. (1) consists of static and dynamic components. The former refers to node embeddings $\{h_i, i \in V\}$ directly attained from the construction encoder. The latter includes features of the solution pool at step t, which is defined as a function $\Phi(P_t) = \{E_k^t, C_k^t\}_{k=1}^K$ that outputs the edges and costs of solutions in P_t.

$$s_t = (\{h_i\}_{i=1}^N, \Phi(P_t)) \tag{1}$$

Action. The action is to construct y complete solutions at step t, i.e., $a_t = \{\delta_1^t, \delta_2^t, ..., \delta_y^t\}$. Note that y depends on the decoder of the construction model. If AM [16] is used, then $y = 1$. For some other construction models that generates multiple solutions with different decoding setting, it is possible that $y > 1$ (e.g., POMO [10], which considers taking each node as the first node of the solutions, generates $y = N$ solutions at each step). In the experiments, we show that FER can improve performance in both of the aforementioned scenarios.

Reward. We record the best-so-far objective value at step t as $C_{\text{bsf}}^t \in \mathbb{R}$, where C_{bsf}^0 is the objective value of the initial solution. The reward is defined as $r_t = C_{\text{bsf}}^{t-1} - C_{\text{bsf}}^t$ which means the decrease in the best-so-far objective value. Note that the reward can also be applied when multiple solutions (i.e., $y > 1$) are constructed at each step. In such case, we independently record y best-so-far objective values of the solutions to calculate the reward.

Transition dynamic. At each step t, we update the solution pool following the designed *diverse solution pool* scheme introduced later in Section IV-D.

IV. Methodology

Given an existing encoder-decoder structured neural construction model, our FER adds a new component called *refiner* between the encoder and the decoder. Intuitively, the refiner aims to leverage more diverse information during search process by dynamically refining the feature embeddings from the encoder. As illustrated in Fig. 2, the refiner uses the sampled historical solutions stored in the diversified pool to iteratively refine the given node embeddings (obtained from the encoder) and improve the diversity and the efficiency for

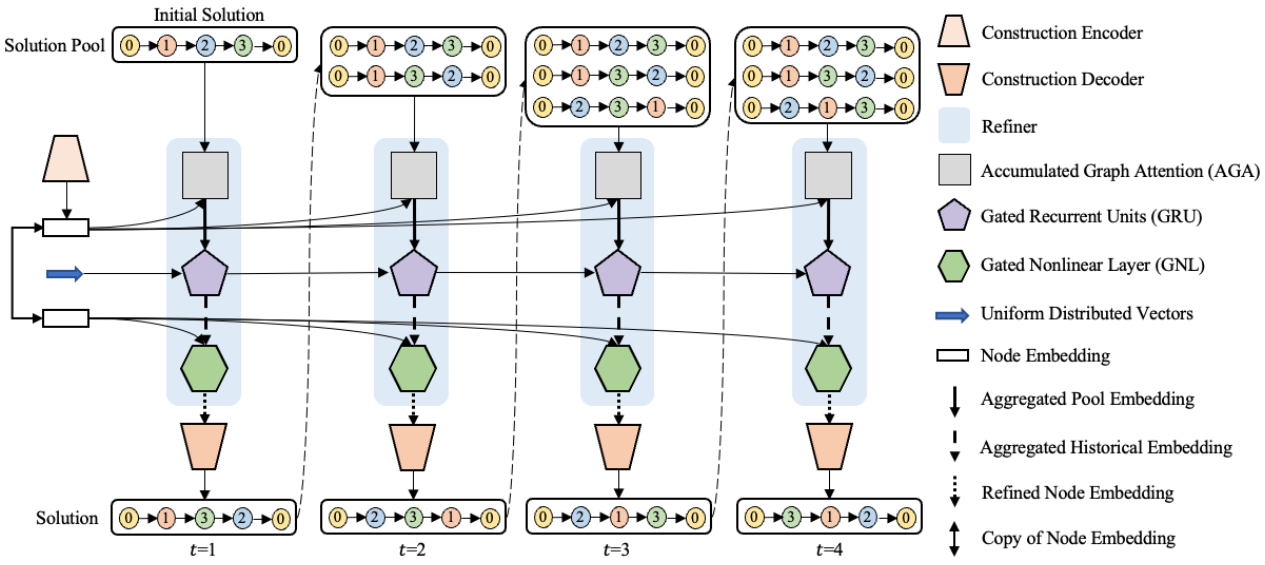


Fig. 2: An illustration of our policy network with solution pool size $K = 3$.

solution reconstruction (performed by the decoder). Specific to the architecture of the refiner network, it mainly comprises an *accumulated graph attention* (AGA) layer, a *gated recurrent units* (GRU) layer and a *gated nonlinear* (GNL) layer. The detailed architecture of these layers in our refiner is displayed in Fig. 3(a). At each step, the AGA first leverages the useful information of historical solutions with their corresponding costs from current pool to derive the aggregated pool embedding. Then the GRU further improves the pool embeddings by fusing more instructive and more global information from previous steps. Afterwards, the GNL adaptively refines the node embeddings from the construction encoder with the guidance of the improved historical embeddings for more exploration. Finally, the refined node embeddings are flowed into the construction decoder for generating solutions, and then the solution pool is updated accordingly. This process is iterated until reaching the step limit T. Below we formally introduce them in detail.

A. The Encoder

In our method, the encoder could be from any pre-trained neural construction model. Here we focus on the Transformer-styled encoder in AM [16] and POMO [10], which delivers state-of-the-art performance. In both AM and POMO, the encoder first embeds problem-specific features to higher-dimensional space, then passes them to stacked attention layers to extract useful information for better representation. Let f_i be the problem-specific features of node $x_i, i \in V$, which contains 2-dimensional location coordinates (for both TSP and CVRP) and 1-dimensional demand vector (for CVRP only). Specifically, f_i is linearly projected to initial node embedding h_i^0 of 128-dimension [16]. Then it is processed through L (3 and 6 for AM and POMO, respectively) attention layers with different parameters to final node embedding h_i^L, where each attention layer is composed of a multi-head attention (MHA) sub-layer and a feed-forward (FF) sub-layer. Following the original design of the Transformer model [4], both the outputs of the MHA sub-layer and the FF sub-layer are followed by a skip-connection layer [29] and a batch normalization (BN) layer [30], as shown in Eq. (2) and Eq. (3), respectively,

$$\tilde{h}_i = \text{BN}(h_i^l + \text{MHA}(h_i^l)), \tag{2}$$

$$h_i^{l+1} = \text{BN}(\tilde{h}_i + \text{FF}(\tilde{h}_i)). \tag{3}$$

MHA sub-layer. The MHA sub-layer uses a multi-head self-attention mechanism [4] with $M = 8$ heads to compute the attention weights between each two nodes. Specifically, the *query/key/value* proposed in [4] are defined with $d_k = d/M$ dimension as shown in Eq. (4).

$$q_i^{l,m} = W_Q^{l,m} h_i^l,\ k_i^{l,m} = W_K^{l,m} h_i^l,\ v_i^{l,m} = W_V^{l,m} h_i^l, \tag{4}$$

Then the attention weights are computed by using the Softmax activation function in Eq. (5) to represent the influence between each two nodes.

$$u_{ij}^{l,m} = \text{softmax}\left(\frac{(q_i^{l,m})^\top (k_i^{l,m})}{\sqrt{d_k}}\right), \tag{5}$$

Finally, the l-th MHA sub-layer first computes the new context vectors by doing an element-wise multiplication of the attention weights with *value* in Eq. (6), and then aggregates the information from M heads in Eq. (7),

$$h_i^{l,m} = \sum_j u_{ij}^{l,m} v_j^{l,m},\ m = 1, 2, ..., M, \tag{6}$$

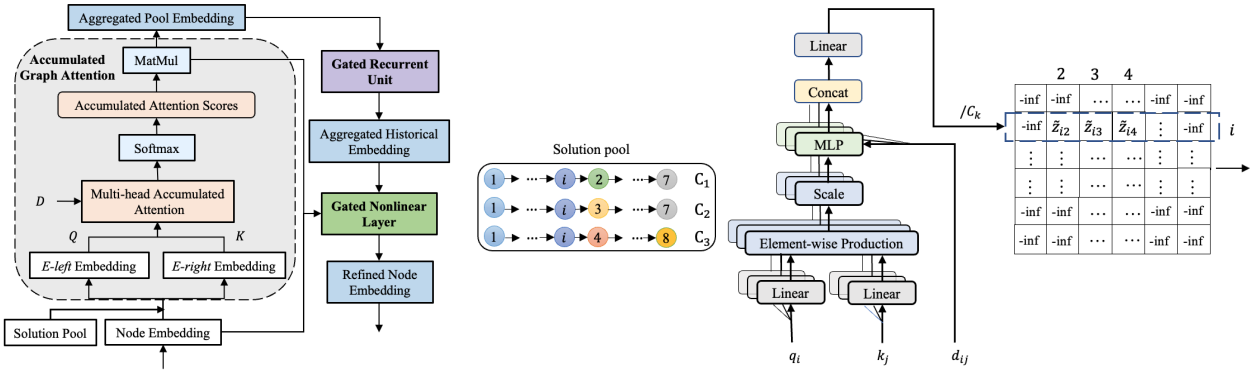
$$\text{MHA}(h_i^l) = [h_i^{l,1}; h_i^{l,2}; ...; h_i^{l,M}] W_O^l, \tag{7}$$

where $W_Q^{l,m}$, $W_K^{l,m}$, $W_V^{l,m} \in \mathbb{R}^{d \times d_k}$, $W_O^l \in \mathbb{R}^{m d_k \times d}$ are learnable parameters, and $[;]$ denotes the concatenate operator.

FF sub-layer. The FF sub-layer processes the node embeddings $\{\tilde{h}_i, i \in V\}$ through a hidden sub-layer with dimension 512 and a ReLU activation function, as shown in Eq. (8),

$$\text{FF}(\tilde{h}_i) = W_{\text{F}}^1\ \text{ReLU}(W_{\text{F}}^0 \tilde{h}_i + b_{\text{F}}^0) + b_{\text{F}}^1, \tag{8}$$

where $W_F^0, W_F^1, b_F^0, b_F^1$ are trainable parameters.



(a) (b)

Fig. 3: Our proposed refiner network for routing problems. (a) The overall architecture; (b) Illustration of the multi-head accumulated graph attention with 3 solutions in the pool.

Finally, the encoder outputs a set of node embeddings in the L-th layer $h_i^L, i \in V$. These embeddings will be preserved as a part of the input to the refiner in each iteration for producing better embeddings, as shown in Fig. 2 and Fig. 3(a).

B. The Refiner

We now elaborate the architecture of the three main layers in our refiner, i.e., the AGA, the GRU and the GNL layer.

AGA layer. To extract the graph topological information from all solutions in the pool $P_t = \{\delta_k^t, C_k^t\}_{k=1}^K$, we design a novel *accumulated graph attention* (AGA) layer based on a proposed multi-head accumulated attention mechanism. Let $e_{ijk}^t \in \{0,1\}$ be a binary variable that indicates whether the edge $\{i,j\}$ exists in the solution δ_k^t at step t. For the edge $\{i,j\}$, we define *E-left* and *E-right* as the node embeddings of its two endpoints, respectively. Our AGA collects *queries* (from *E-left*) and *keys* (from *E-right*) to perform the multi-head accumulated graph attention as shown in Fig. 3(b). Given the node embeddings $\{h_i, h_j\}$, the multi-head accumulated attention first linearly projects the node embeddings to form *queries* ($q_i = h_i w_1^m$) and *keys* ($k_j = h_j w_2^m$) in the M heads of dimension d_k, where $w_1^m, w_2^m \in \mathbb{R}^{d \times d_k}$ are learnable parameters. Then it processes them via an element-wise production and concatenates the output with the distance d_{ij} of the edge $\{i,j\}$ to compute initial weights, which are further fine-tuned through a Feed-Forward Tuning (FFT) sub-layer[1] to generate the fused weights w_{ijk}^t by gathering information from M heads. Note that the weight of the edge $\{i,j\}$ equals to 0 if it does not exists in P_t. In doing so, the network could automatically learn the best fusion with multiple heads from different perspectives. Below, we summarize the formulation of the above process as Eq. (9),

$$a_{ijk}^m = \begin{cases} \text{FFT}\left(\left[\frac{q_i \cdot k_j}{\sqrt{d_k}}; d_{ij}\right]\right), & \text{if } e_{ijk}^t = 1 \\ 0, & \text{otherwise} \end{cases} \qquad (9)$$
$$w_{ijk}^t = [a_{ijk}^1; a_{ijk}^2; ...; a_{ijk}^M] w_o,$$

where $w_o \in \mathbb{R}^{M \times 1}$ are learnable parameters.

[1]The FFT includes a feed-forward network with 3-layers (dimensions are 2, 16 and 1, respectively) and a ReLU activation function.

Based on the fused weights, we define a matrix $\tilde{Z}^t \in \mathbb{R}^{N \times N}$, where each element $\tilde{z}_{ij}^t$ (initialized as negative infinity) stores the accumulated compatibility weights of the corresponding edge $\{i,j\}$. More specifically, $\tilde{z}_{ij}^t$ defined in Eq. (10) is accumulated by summing up the fused weights of the edge $\{i,j\}$ (after being scaled by the corresponding cost of the solution) appearing in all solutions in the current pool,

$$\tilde{z}_{ij}^t = \sum_{k=1}^{K} e_{ijk}^t w_{ijk}^t / C_k^t. \qquad (10)$$

The logic of dividing the weight by C_k^t (i.e., the solution cost δ_k^t) is that solutions of higher quality in the pool should have greater impacts. In doing so, the accumulated compatibility vector $\tilde{z}_i^t = \{\tilde{z}_{i1}^t, ..., \tilde{z}_{iN}^t\}$ stores all edge information concerning node i in the pool at step t. Specifically, only the elements which refer to the edges starting from the node i in the pool are not negative infinity. In Eq. (11), the accumulated compatibility $\bar{Z}^t \in \mathbb{R}^{N \times N}$ is then normalized by the softmax function to calculate the accumulated attention weights,

$$Z^t = \text{softmax}(\tilde{Z}^t). \qquad (11)$$

The aggregated pool embedding of node i is further gathered as the linear combination of the node embeddings of its successors across the whole solution pool, as shown in Eq. (12),

$$\bar{h}_i^t = \sum_j z_{ij}^t h_j. \qquad (12)$$

Note that AGA layer could not be removed from the *refiner* since it is a basic component to provide diverse information (aggregated pool embeddings) learned from dynamic historical solutions stored in the pool. Based on AGA, subsequent components can further learn to improve the aggregated pool embeddings (via GRU cell) and refine the node embeddings from the encoder (via GNL layer).

GRU layer. The aggregated pool embedding of each node i is further processed using a gated recurrent unit (GRU) cell to extract and memorize global historical information from all the previous steps, and the output is expressed in Eq. (13),

$$\breve{h}_i^t = \text{GRUcell}(\breve{h}_i^{t-1}, \bar{h}_i^t). \qquad (13)$$

GNL layer. Given the aggregated historical embedding $\breve{h}_i^t, i \in V$ at step t, we design a novel *gated nonlinear* (GNL) layer to refine the node embeddings $h_i, i \in V$ generated from the construction encoder, which is fixed and shared across the whole process. We leverage an *absorb* gate to absorb the new guidance information from the aggregated historical embedding at the current step, and a *reserve* gate to reserve the desirable property of node embeddings, i.e., $u_t = \sigma(W_u[h_i; \breve{h}_i^t])$ and $r_t = \sigma(W_r[h_i; \breve{h}_i^t])$, where W_u and W_r are trainable parameters. The two gates combine the information taken from the aggregated historical embeddings and the given node embeddings. Then a nonlinear active function is used to express absorbed new information from current pool, as shown in Eq. (14),

$$\hat{h}_i^t = \tanh(W[h_i; u_t \cdot \breve{h}_i^t]),\ i \in V, \tag{14}$$

where W is trainable parameters. In Eq. (15), the final node embeddings $\vec{h}_i^t, i \in V$ are defined as the linear combination of new guidance information and original node embeddings,

$$\vec{h}_i^t = r_t \cdot h_i + (1 - r_t) \cdot \hat{h}_i^t,\ i \in V. \tag{15}$$

The function of each component of the proposed *refiner* is verified in Section V-D.

C. The Decoder

Similar to the encoder, the decoder could also be from any pre-trained neural construction model. We still apply the the Transformer-styled decoder in AM and POMO. However, instead of taking the original embeddings h_i generated by the encoder, the decoder in our model takes the final node embeddings $\vec{h}_i, i \in V$ from the refiner as input, based on which it sequentially selects a node at each decoding step to sample a complete solution $\delta = \{\pi_1, \pi_2, ...\pi_Z\}$, where Z refers to decoding steps. In specific, Z equals to N for TSP but may be larger than N for CVRP due to multiple visits to the depot.

Taking TSP as an example, the decoder first calculates the mean of node embeddings to provide a more global perspective, i.e., $\dot{h}^t = \frac{1}{N}\sum_i^N \vec{h}_i^t$, then defines a context vector in Eq. (16) as the combination of the mean graph embedding, the embeddings of the end node of the route and the last visited node (we omit step t for simplification),

$$h^c = (\dot{h}, \vec{h}_{\pi_1}, \vec{h}_{\pi_{z-1}}), \tag{16}$$

where $\vec{h}_{\pi_z}$ is the node embedding of the node visited at decoding step z. For the first step, the last two elements of h^c are replaced by trainable parameters. The context vector and refined node embeddings are then processed by a MHA layer as introduced in Section IV-A to generate a glimpse vector h^g in Eq. (17),

$$h^g = \text{MHA}(W_Q^g h^c,\ W_K^g \vec{h},\ W_V^g \vec{h}). \tag{17}$$

after which the decoder computes the compatibility between the enhanced glimpse and node embeddings, and further the probability of selecting the next node to visit at decoding step z as shown in Eq. (18) - (19),

$$c^z = G \cdot \tanh\left(\frac{(h^g W_Q)^T (\vec{h} W_K)}{\sqrt{d_k}}\right), \tag{18}$$

$$p^z = \text{Softmax}(c^z), \tag{19}$$

where $W_Q^g, W_K^g, W_V^g, W_Q, W_K$ are trainable parameter matrices, and G is often set to 10 to control the entropy of c^z.

Regarding POMO, since it constructs N solutions by taking each of the N nodes as the first node to visit (i.e., $\vec{h}_{\pi_1}^i = \vec{h}_i$), it defines N context vectors $h_i^c, \forall i \in V$ as shown in Eq. (20),

$$h_i^c = \begin{cases} (\dot{h}, \vec{h}_i, \vec{h}_{\pi_{z-1}}^i), & \text{if } z > 1 \\ \text{None}, & \text{if } z = 1 \end{cases} \tag{20}$$

The N context embeddings are then processed following Eq. (17)-(19) in parallel to obtain the probability of picking the next node in each of the N solutions.

Pertaining to the decoding strategy, we could choose the node with the maximum probability in a *greedy* manner or sample a node according to the probability in a *sampling* manner at each decoding step. The newly generated solutions will be added to the pool P_t following the pool updating mechanism introduced in Section IV-D.

D. Solution Pool Diversification

Solution diversity is important for the search performance. We design the following mechanism to increase the solution diversity of our FER. First, we randomly replace a solution δ_i from the current pool with the newly constructed one δ_t (by the decoder). Second, with a probability gradually increased as the training progresses, we further choose to replace another solution δ_j in the pool with a randomly generated new one δ_t^R following the rules below: we first check whether the current pool contains a randomly generated solution added in the previous steps; if yes, such solution is set to δ_j; if not, we randomly choose another solution from the pool as δ_j ($\delta_j \neq \delta_t$). We summarize the above operation in Eq. (22),

$$P_t = \begin{cases} P_{t-1} \setminus \{\delta_i, \delta_j\} \cup \{\delta_t, \delta_t^R\}, & \text{if } \epsilon < \exp(\frac{elnE}{E}) \\ P_{t-1} \setminus \{\delta_i\} \cup \{\delta_t\}, & \text{otherwise} \end{cases} \tag{21}$$

where $i \neq j$, "$\setminus$" and "$\cup$" are the set subtraction and union operator, respectively. The decision probability is computed based on the epoch index e, the total epoch number E (defined in the below subsection), and a random number $\epsilon \in [1, E+1]$. For POMO, we randomly select K solutions from the newly sampled N solutions to update the pool.

In doing so, the pool could, 1) provide a more diverse dynamic state (in MDP) with different solutions between step $t-1$ and t; 2) achieve a better balance between exploration and exploitation by gradually converging to the case that only one random solution is contained in the pool.

E. Training Algorithm

As summarized in Algorithm 1, we adopt n-step advantage actor critic (A2C) algorithm to train our FER to optimize

TABLE I: Gaps of FER-AM with different pool sizes.

Pool Size	N=20			N=50			N=100		
	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
K=2	6.165	0.11%	29m	10.518	0.23%	1.6h	16.025	0.33%	5.7h
K=4	6.163	0.07%	30m	10.498	0.04%	1.8h	16.010	0.23%	6.7h
K=6	6.159	0.00%	32m	10.495	0.01%	2h	15.988	0.09%	7.4h
K=8	6.159	0.00%	34m	10.495	0.01%	2.3h	15.974	0.01%	8h
K=10	6.159	-	36m	10.494	-	2.8h	15.973	-	9.1h

Note: We use Obj. as the abbreviation for the objective value throughout the paper.

ALGORITHM 1: n-step A2C

Input: construction encoder π_e with pre-trained parameters θ_e (fixed); *refiner* π_r with randomly initialized parameters θ_r (trainable); construction decoder π_d with pre-trained parameter θ_d (trainable); critic v_ϕ with randomly initialized parameters ϕ; learning rate β_π, β_ϕ;
foreach $e = 1, 2, ..., E$ **do**
  foreach $b = 1, 2, ..., B$ **do**
    Randomly generate training instances I_b; $t \leftarrow 0$;
    Get node embeddings using pre-trained construction encoder, $h = \pi_e(I_b|\theta_e)$, as static component of s_t;
    while $t < T$ **do**
      $d\theta_{r,d} \leftarrow 0$; $d\phi \leftarrow 0$;
      while $t - t_s < n_{step}$ *and* $t < T$ **do**
        Perform the refiner to get $\vec{h}^t = \pi_r(\vec{h}^t|s_t, \theta_r)$;
        Sample solutions, i.e., $a_t \sim \pi_d(a_t|\vec{h}^t, \theta_d)$;
        Get reward r_t and the next state s_{t+1}. $t \leftarrow t + 1$;
      end
      Bootstrap from the last state $R = v_\phi(s_t)$;
      foreach $i = t - 1, ..., t_s$ **do**
        $R \leftarrow r_i + \gamma R$;
        Accumulate gradients: $d\theta_{r,d} \leftarrow d\theta_{r,d} + \frac{1}{|I_b|y} \sum_{I_b} \sum_y (R - v_\phi(s_i)) \nabla log \pi_{\theta_{r,d}}(a_i|s_i)$;
        Accumulate gradients: $d\phi \leftarrow d\phi + \frac{1}{|I_b|y} \sum_{I_b} \sum_y (R - v_\phi(s_i))\ \nabla v_\phi(s_i)$;
      end
      $\theta_r \leftarrow \theta_r + \frac{\beta_\pi}{n_{\text{step}}} d\theta_r$; $\theta_d \leftarrow \theta_d + \frac{\beta_\pi}{n_{\text{step}}} d\theta_d$;
      $\phi \leftarrow \phi - \frac{\beta_\phi}{n_{\text{step}}} d\phi$.
    end
  end
end

the objective value defined in Section III, which performs E epochs of training. The actor $\{\pi_e, \pi_d, \pi_r\}$ consists of the pre-trained construction encoder π_e and decoder π_d, and the introduced *refiner* π_r, where the parameters of the encoder π_e are fixed during training. Regarding the critic network v_ϕ, it takes the refined node embeddings generated by the *refiner* as the input and computes the estimated state value. The structure of the critic network is similar to [13], which first concatenates the refined node embeddings and its mean pooling as the fused ones, and then process the fused embeddings by a multilayer perception (MLP) layer [31], [32] to obtain the output value.

V. Experiments

We conduct experiments on two most widely studied VRPs, i.e., TSP and CVRP, to verify the effectiveness of our method. Following [10], [15], [16], we randomly generate instances with N (20, 50 and 100) nodes for each problem for training and use the same 10,000 instances from [16] for testing. To evaluate the applicability of our FER to different neural construction methods, we apply it to two prevailing deep models, i.e., AM[2] and POMO[3], by adopting their original encoder and decoder. We call the corresponding new models as *FER-AM* and *FER-POMO*, respectively.

TABLE II: Obj. of FER-AM with different pool schemes.

Pool Scheme	CVRP20	CVRP50	CVRP100
Greedy	6.186	10.569	16.116
Diversified	6.181	10.560	16.097

We train our FER with $B = 20$ batches per epoch for $E = 100$ epochs for CVRP20 and CVRP50 and $E = 200$ epochs for CVRP100. Regarding FER-AM, we set batch size to 512 and $n_{step} = 4$ with $T = 100$ steps. The Adam Optimizer [33] is adopted with initial learning rate 10^{-4} for actor and 5×10^{-5} for critic with decay rate equal to 0.99. Regarding FER-POMO, we set batch size to 32 and $n_{step}=5$ with $T=100$ steps. The initial learning rate is set to 5×10^{-5} for actor and 10^{-6} for critic with decay rate equal to 0.988. The decoder follows the design of the corresponding original construction methods, which samples a single solution for AM and N solutions for POMO at each step. Our dataset and code in Pytorch are available[4].

A. Solution Pool Analysis

We first analyze the impacts of the solution pool size and the solution pool diversification scheme in our proposed FER.

Impacts of the solution pool size. In Table I, we train and test our FER-AM using different numbers of K (from 2 to 10) for solving CVRP instances with $T = 5,000$ to show the impacts of the solution pool size. The gaps are calculated based on the solutions obtained by the method with lowest objective values (i.e., total length of the solutions), and the time refers to the computation time of solving all 10,000 instance for all methods. We can observe that larger pool sizes tend to achieve lower objective values and optimality gaps but longer computation time. To achieve a desirable trade-off between the solution quality and the computation cost, we adopt $K = 6$ for CVRP20 and CVRP50, and $K = 8$ for CVRP100 for both FER-AM and FER-POMO in the following experiments.

Impacts of the solution pool diversification scheme. In Table II, we replace the solution pool P_t with a greedy one which

[2]https://github.com/wouterkool/attention-learn-to-route
[3]https://github.com/yd-kwon/POMO
[4]https://github.com/Demon0312/Feature-Embedding-Refiner

TABLE III: Comparison with various baselines on TSP and CVRP.

	Method	N=20 Obj.	N=20 Gap	N=20 Time#	N=50 Obj.	N=50 Gap	N=50 Time#	N=100 Obj.	N=100 Gap	N=100 Time#
TSP	Gurobi	3.84	-		5.70	-		7.76	-	
	LKH	3.84	0.00%	(8m)	5.70	0.00%	(2h)	7.76	0.00%	(8h)
	AM ($S=200$)	3.84	0.11%	(30s)	5.73	0.59%	(2m)	7.97	2.63%	(10m)
	FER-AM ($T=200$)	3.84	0.03%	(55s)	5.71	0.18%	(3m)	7.90	1.64%	(12m)
	AM ($S=5,000$)	3.84	0.06%	(10m)	5.72	0.38%	(45m)	7.93	2.13%	(4.3h)
	FER-AM ($T=5,000$)	3.84	0.01%	(13m)	5.70	0.02%	(57m)	7.85	1.12%	(5h)
	AM+LCP‡ (1,280, 10)		-		5.70	0.08%	(4.3h)	7.85	1.13%	(8.8h)
	POMO ($S=200$)	3.84	0.02%	(2m)	5.70	0.06%	(12m)	7.79	0.27%	(56m)
	FER-POMO ($T=200$)	3.84	0.00%	(3m)	5.70	0.03%	(14m)	7.78	0.15%	(1.1h)
	POMO ($S=200$, $\times 8$ augment)	3.84	0.00%	(18m)	5.70	0.01%	(1.5h)	7.77	0.08%	(7.4h)
	FER-POMO ($T=200$, $\times 8$ augment)	3.84	**0.00%**	(24m)	5.70	**0.00%**	(1.8h)	7.77	**0.03%**	(8.5h)
CVRP	HGS	6.13	-	(12.5h)	10.37	-	(24.5h)	15.56	-	(56h)
	LKH	6.14	0.16%	(22h)	10.38	0.14%	(3.6d)	15.65	0.56%	(6.5d)
	AM ($S=200$)	6.27	2.28%	(1m)	10.67	2.93%	(4m)	16.29	4.67%	(14m)
	FER-AM ($T=200$)	6.18	0.82%	(2m)	10.56	1.87%	(6m)	16.10	3.43%	(22m)
	AM ($S=5,000$)	6.24	1.80%	(25m)	10.60	2.26%	(1.5h)	16.15	3.77%	(5.5h)
	FER-AM ($T=5,000$)	6.16	0.50%	(32m)	10.50	1.24%	(2h)	15.97	2.64%	(8h)
	AM+LCP‡ (2,560, 1)	6.15	0.35%		10.52	1.49%		16.00	2.81%	
	AM+LCP‡ (6,500, 1)		-			-		15.98	2.68%	
	POMO ($S=200$)	6.15	0.33%	(3m)	10.44	0.71%	(20m)	15.74	1.14%	(1.6h)
	FER-POMO ($T=200$)	6.13	0.04%	(4m)	10.41	0.42%	(35m)	15.70	0.88%	(1.8h)
	POMO ($S=200$, $\times 8$ augment)	6.14	0.17%	(22m)	10.40	0.33%	(2.6h)	15.67	0.01%	(12h)
	FER-POMO ($T=200$, $\times 8$ augment)	6.13	**0.01%**	(28m)	10.38	**0.16%**	(3h)	15.65	**0.56%**	(14h)

Note: The gaps are calculated based on the solutions obtained by Gurobi for TSP and HGS for CVRP.
The time refers to the computation time of solving all 10,000 instance for all methods.
‡ LCP improves the vanilla AM for better performance and is denoted as "AM+LCP" according to its original paper. We run the results of TSP using the pre-trained models and take the objective values and optimality gaps of CVRP from the original paper since the pre-trained CVRP models are not provided.

reserves the best K solutions for solving CVRP with $T=200$. We can observe that the diversified pool consistently achieves lower objective values than the greedy one, and the superiority of our diversified pool is more obvious as the problem scales up, which verifies the effectiveness of our design.

B. Comparison Analysis

We now compare FER with neural construction models, i.e., AM and POMO, to show the effectiveness of our method. Regarding AM, we use its sampling strategy to generate S solutions to solve an instance, where S is set to 200 and 5,000, respectively. Regarding POMO with diverse rollout strategy, we use sampling strategy with $S = 200$ with and without $\times 8$ augment to solve an instance. It generates $8 \times 200 \times N$ and $200 \times N$ solutions, respectively, where N is the size of an instance (number of nodes). For fair comparison, we test FER-AM and FER-POMO by sampling the same number of solutions to AM and POMO, respectively. We also consider the most recent neural model LCP [28][5] as a baseline, which combines both construction and improvement methods to refine AM. To calculate the optimality gaps for all neural models, we leverage three strong conventional solvers, i.e., Gurobi [34][6], LKH [35][7], and HGS [36][8] (the state-of-the-art conventional CVRP solver). Note that it is hard to absolutely fair compare the run time between neural methods (Python, GPU) and conventional solvers (C++, CPU), thus we follow the conventions to report the total run time using a single TITAN XP GPU (for neural methods) or a single CPU core at 2.0 GHz (for conventional methods).

The comparison results for TSP and CVRP are summarized in Table III. We can observe that our FER-AM significantly improves the performance of AM on both TSP and CVRP, despite slightly longer computation time. Our FER-AM ($T=200$) outperforms both AM ($S=200$) and AM ($S=5,000$) for all cases in terms of objective values and gaps, and consumes shorter computation time than AM ($S = 5,000$). With step limit $T=5,000$, FER-AM achieves slightly better performance than AM+LCP with much lower computational costs.

Pertaining to the state-of-the-art method POMO, our FER can still improve its performance in terms of the gap. With step limit $T = 200$, our FER-POMO outstrips POMO ($S = 200$) for all cases with slightly longer computation time. It also outperforms POMO ($S = 200$, $\times 8$ augment) on TSP20 and CVRP20 with much shorter computation time, and delivers competitive results for larger problem sizes. Furthermore, FER-POMO ($T = 200$) even exhibits superior performance to the specialized heuristic solver LKH and almost the same performance to the state-of-the-art HGS on CVRP20. By leveraging the same data augmentation strategy in POMO, FER-POMO ($T=200$, $\times 8$ augment) further improves the solution

[5]https://github.com/alstn12088/LCP
[6]https://www.gurobi.com/
[7]http://webhotel4.ruc.dk/∼keld/research/LKH-3/
[8]https://github.com/vidalt/HGS-CVRP/tree/main/Program

qualities and delivers better results than POMO ($S = 200$, $\times 8$ augment) for all cases. In particular, our FER-POMO ($T = 200$, $\times 8$ augment) achieves the lowest objective values and gaps among all the neural heuristics, and exhibits almost the same performance to LKH and slightly worse performance than HGS. The superiority of our method well justified the effectiveness of the proposed encoder-refiner-decoder structure, which allows more efficient search for neural construction models to produce higher quality solutions.

Furthermore, the existing evolutionary algorithms for VRPs also maintain a solution pool (a population) for crossover and mutation to improve the solution quality, which shares some similarities to our FER method. We thus compare our FER with the population-based evolutionary algorithm, e.g., genetic algorithm (GA) [37] on CVRP with 1,000 instances as shown in Table IV, where the gaps are calculated based on the solutions obtained by the method with lowest objective values. Specifically, we take the initial solution pool of our FER-AM as the initial population of GA and run two methods with the same iterations, i.e., 200 and 5,000. For example, GA (FER-AM Pop, *iter*=200) refers to GA with the same initial solution pool as FER-AM (T=200). The first row refers to GA following the settings in the original paper, where we further add more iterations for better performance. From Table IV, we can observe that with the same initial solution pool (population) and iterations, our FER-AM outperforms GA in terms of the objective values and optimality gaps on all cases with less computation time. With larger population size and iterations, GA achieves lower optimality gaps with longer computation time. However, it is still inferior to our FER-AM, which shows the effectiveness of our method.

C. Efficiency Analysis

We continue to compare the search efficiency of our FER with three representitive neural *improvement* methods, i.e., Wu et al. [13][9], NLNS [24][10](CVRP only), and DACT [15][11] (state-of-the-art) for solving the instances of TSP50, TSP100, CVRP50 and CVRP100, respectively. We use the pre-trained models for these improvement methods which are available online. Regarding DACT, we use the same $\times 8$ data augments as FER-POMO ($S = 200, \times 8$ augment) with $T = 5,000$ iterative steps similar to other improvement methods, so as to largely preserve its favorable performance.

The curves of searching progress for these methods are plotted in Fig. 4, where the horizontal coordinate refers to the iterative steps and the vertical one refers to the best-so-far objective values averaged over 10,000 instances used in Table III. Here, $T = 5,000$ was used expect for FER-POMO where $T = 200$ was used. We further plot the best objective values of the vanilla AM ($S = 5,000$) in blue dotted lines and that of the vanilla POMO ($S = 200, \times 8$ augment) in green dotted lines as baselines to show the improvement of our FER. We can observe the large gaps between AM and FER-AM, which verifies the superiority of FER. Although FER-AM is inferior to POMO, which indicates that the performance of our FER depends on the performance of pre-trained backbone models, it is a fair case since one approach (like our FER) can not guarantee to boost the backbone model to outperform other more advanced and larger models (e.g., POMO[12]). Furthermore, although the improvement of FER-POMO to the state-of-the-art POMO is less significant when compared to the case of FER-AM, it is also reasonable since it is really hard to significantly improve a highly-optimized model. Nevertheless, our FER can still improve its performance and enhance its generalization ability (refer to Table VI and Table VII), which shows the effectiveness of our method. For *improvement* baselines, it can be observed that our FER-AM converges much faster than Wu et al. [13] and achieves lower objective values for all cases. With comparable performance to NLNS, our FER-AM converges faster and outperforms NLNS given limited iterative steps (i.e., 1,000 steps). Regarding our FER-POMO ($S = 200$, $\times 8$ augment), it significantly outstrips both Wu et al. [13] and NLNS for all cases where the superiority is more salient on larger sizes. More importantly, it also converges much faster than the state-of-the-art neural improvement method DACT ($T = 5,000$, $\times 8$ augment) for all cases, which further verifies the significance of our method.

D. Further Analysis of our FER

We now provide more analysis to study the effects of the proposed *refiner* network and its components.

Effect of the *refiner*. To verify that the *refiner* can effectively refine feature embeddings from the encoder and further contribute to diverse exploration, we visualize the probability distributions of selecting the next node during solving a CVRP50 instance in Fig. 5. We perform 200 steps of searching using FER-POMO with and without the introduced refiner, respectively, and produce one solution at each step for demonstration. Given a fixed partial solution shown in the left side of Fig. 5, we plot the probability distributions of selecting the next node at the 50th, 100th, 150th, and 200th steps for FER-POMO and FER-POMO-w/o-Refiner, respectively. We find that the probability of selecting the next node for FER-POMO-w/o-Refiner is always the same through the whole searching process, and the sole diversity in solutions comes from sampling with the same distribution. On the other hand, our FER-POMO could explore more diverse solutions by consistently refining the node embeddings and gradually update the probability distributions for sampling, which helps improve the solution quality (objective values at the 200th step are 11.53 and 11.70 for FER-POMO and FER-POMO-w/o-Refiner, respectively) and the sampling efficiency.

Effect of each component of the *refiner*. In Table V, we conduct an ablation study to showcase the effect of each part of the refiner on CVRP50 instances using FER-AM, where the gaps are calculated based on the solutions obtained by the method with lowest objective values. The marker "✓" and "×"

[9]https://github.com/WXY1427/Learn-Improvement-Heuristics-for-Routing

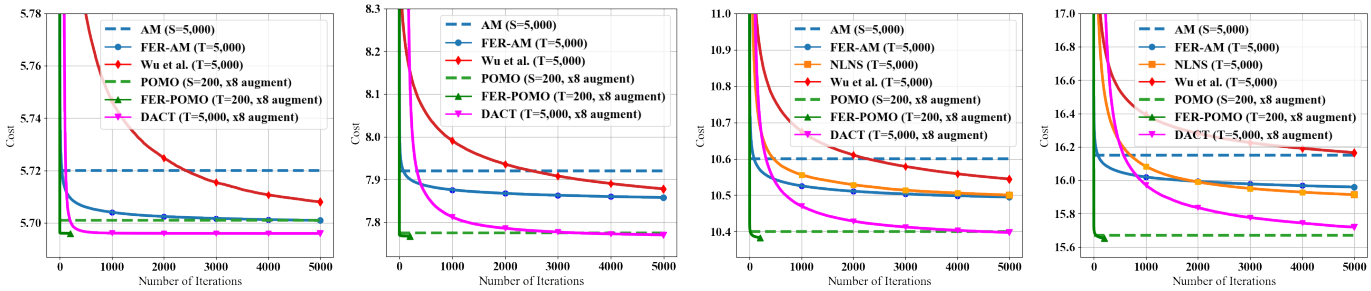
[10]https://github.com/ahottung/NLNS

[11]https://github.com/yining043/VRP-DACT

[12]POMO leverages group baselines during training and the data augmentation schemes during inference, and the number of parameters of POMO is almost twice that of AM.

TABLE IV: Comparison with GA on 1,000 CVRP instances.

Pool Size	N=20 Obj.	N=20 Gap	N=20 Time	N=50 Obj.	N=50 Gap	N=50 Time	N=100 Obj.	N=100 Gap	N=100 Time
GA (Pop-Size=50, *iter*=12,000)	6.251	0.95%	(14h)	10.781	1.88%	(46h)	16.392	2.35%	(4d)
GA (FER-AM Pop, *iter*=200)	6.357	2.67%	(3m)	10.991	3.87%	(8m)	16.723	4.42%	(14m)
FER-AM (T=200)	6.217	0.40%	(23s)	10.653	0.67%	(1.8m)	16.141	0.79%	(2.7m)
GA (FER-AM Pop, *iter*=5,000)	6.278	1.39%	(1.2h)	10.889	2.90%	(3h)	16.533	3.24%	(5.8h)
FER-AM (T=5,000)	6.192	-	(10m)	10.582	-	(47m)	16.015	-	(1.1h)



(a) TSP50 (b) TSP100 (c) CVRP50 (d) CVRP100

Fig. 4: Curves of searching progress for our FER and baselines including AM [16], POMO [10], NLNS [24], Wu et al. [13], and DACT [15].

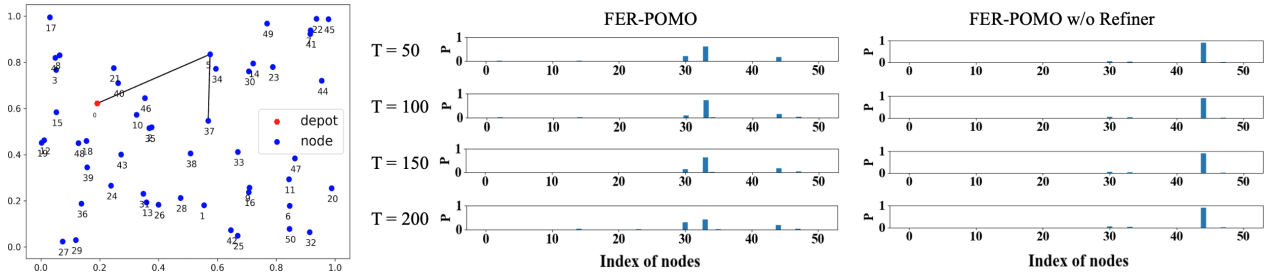


Fig. 5: Probability distributions of FER-POMO and POMO for taking actions at last visited node 37 given a fixed partial solution $\{0, 5, 37\}$. The distributions at four different steps are presented, i.e., 50, 100, 150 and 200.

TABLE V: Effects of each component of the *refiner*.

AGA	GRU	GNL	Obj.	Gap	Time
✓	×	×(+)	10.605	1.05%	1.8h
✓	✓	×(+)	10.518	0.22%	1.9h
✓	×	✓	10.512	0.16%	1.9h
✓	✓	✓	10.495	-	2h

refer to use the corresponding part or not, respectively, and the marker "×(+)" refers to replace GNL layer with simply adding the fixed node embeddings from the encoder and the refined embeddings from previous components together. Please note that the AGA layer could not be removed from FER since it is a basic component of the *refiner* to provide diverse information as introduced in Section IV-B. Thus AGA is kept for the last four rows. From Table V, we can observe that GNL layer outperforms "(+)" strategy in terms of the objective values and optimality gaps for the method with and without GRU cell, since GNL layer could adaptively combine the pool embeddings and the node embeddings for more desirable representation. Moreover, GRU cell can also improve the performance for the method with both GNL layer and "(+)" strategy since it aggregates the instructive and global information learned from AGA in all previous steps. Further combining GRU and GNL together, our FER-AM (the final row) achieves the best performance in terms of the objective values and optimality gaps.

E. Generalization Analysis

We now evaluate the generalization performance of our FER on two well-known benchmarks, i.e., TSPLIB [38][13] and CVRPLib [39][14], respectively. We compare FER-POMO ($T = 200, \times 8$ augment) with Wu et al. [13] and POMO ($S = 200, \times 8$ augment), and record the results in Table VI and Table VII. Similar to Wu et al. [13], we use the models trained for TSP100 and CVRP100 to solve those instances, with sizes from 51 to 200 and 101 to 200, respectively. For POMO and FER-POMO, we construct solutions by sampling a node from candidates with three highest probabilities for

[13]http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[14]http://vrp.atd-lab.inf.puc-rio.br/index.php/en/

TABLE VI: Generalization on TSPLIB instances.

Instance	Opt.	Wu et al. [13] ($T=3000$)	POMO ($T=200$)	FER-POMO ($T=200$)
eil51	425	438	**429**	**429**
berlin52	7,544	8,020	**7,544**	**7,544**
st70	675	706	**677**	**677**
eil76	538	575	545	**544**
pr76	108,159	**109,668**	120,404	116,337
rat99	1,211	1,419	1,268	**1,266**
KroA100	21,282	25,196	21,893	**21,720**
KroB100	22,141	26,563	23,044	**23,043**
KroC100	20,749	25,343	21,475	**21,372**
KroD100	21,294	24,771	22,387	**21,969**
KroE100	22,068	26,903	22,787	**22,679**
rd100	7,910	7,915	**7,910**	**7,910**
eil101	629	658	**640**	**640**
lin105	14,379	18,194	16,248	**16,185**
pr107	44,304	53,056	45,946	**44,482**
pr124	59,030	66,010	**59,031**	**59,031**
bier127	118,282	142,707	121,037	**120,355**
ch130	6,110	7,120	**6,111**	**6,111**
pr136	96,772	105,618	97,434	**97,215**
pr144	58,537	71,006	**59,358**	59,463
ch150	6,528	7,916	6,566	**6,563**
KroA150	26,524	31,244	29,628	**29,190**
KroB150	26,130	31,407	28,829	**28,419**
pr152	73,682	85,616	**75,920**	76,046
u159	42,080	51,327	**42,717**	42,801
rat195	2,323	**2,913**	3,129	3,013
d198	15,780	**17,962**	22,305	21,769
KroA200	29,368	35,958	35,440	**35,283**
KroB200	29,437	36,412	35,518	**35,382**
Avg.Gap	0.00%	15.56%	7.07%	**6.25% (↓13.12%)**

Bold refers to the best performance among all learning based methods. Opt. is the abbreviation for Optimal (or the best known) objective value, which is available in the public benchmark library.

TABLE VII: Generalization on CVRPLIB instances.

Instance	Opt.	Wu et al. [13] ($T=5,000$)	POMO ($T=200$)	FER-POMO ($T=200$)
X-n101-k25	27,591	29,716	28,000	**27,910**
X-n106-k14	26,362	27,642	**26,570**	26,617
X-n110-k13	14,971	15,927	15,111	**14,986**
X-n115-k10	12,747	14,445	12,888	**12,816**
X-n120-k6	13,332	15,486	13,476	**13,442**
X-n125-k30	55,539	60,423	57,914	**57,569**
X-n129-k18	28,940	32,126	29,094	**29,051**
X-n134-k13	10,916	12,669	11,260	**11,219**
X-n139-k10	13,590	15,627	13,763	**13,730**
X-n143-k7	15,700	18,872	**15,900**	15,903
X-n148-k46	43,448	50,563	49,191	**48,218**
X-n153-k22	21,220	26,088	**23,593**	23,650
X-n157-k13	16,876	19,771	17,195	**17,144**
X-n162-k11	14,138	16,847	14,562	**14,505**
X-n167-k10	20,557	24,365	21,136	**21,089**
X-n172-k51	45,607	**51,108**	54,060	52,443
X-n176-k26	47,812	57,131	51,610	**50,930**
X-n181-k23	25,569	27,173	25,975	**25,953**
X-n186-k15	24,145	28,422	24,768	**24,639**
X-n190-k8	16,980	20,145	18,545	**18,412**
X-n195-k51	44,225	51,763	47,390	**46,640**
X-n200-k36	58,578	64,200	60,725	**60,353**
Avg.Gap	0.00%	14.27%	4.53%	**3.78% (↓19.82%)**

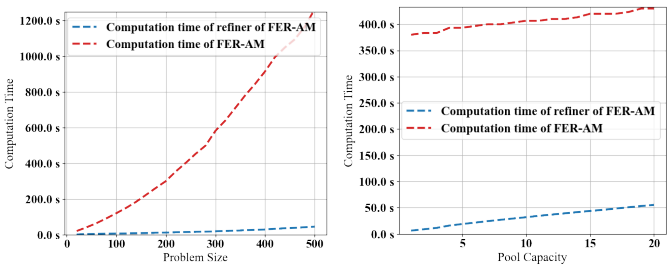
more stable inference on both TSPLIB and CVRPLIB, which have different distributions of customer locations and larger problem sizes in comparison with the training instances. We report the objective value to each instance and also the average gaps based on the optimal solutions provided in the datasets. In Table VI, we can observe that FER-POMO significantly outperforms Wu et al. [13] and POMO in terms of the average gap. Especially, though POMO produces high-quality solutions, FER-POMO still could reduce its gap by **13.12%** on TSPLIB. Moreover, FER-POMO achieves the lowest objective values among neural methods on most instances of various scales, which shows stable superiority of our method even on large-scale instances. In Table VII, patterns similar to that of TSPLIB could be observed, where our FER-POMO outstrips all other neural methods on CVRPLIB and improves the performance of POMO by **19.82%**. The superiority in generalization performance further justifies the effectiveness of our encoder-refiner-decoder structure which allows more diverse search to improve the solution quality.

F. Complexity Analysis

We finally provide complexity analysis of the proposed FER, especially the designed *refiner*. We further plot the computation time of FER-AM and its *refiner* part over 1) problem size from 20 to 500 with K=2 in Fig. 6(a) and 2) pool size from 2 to 20 with N=100 in Fig. 6(b) on 1,000 randomly generated instances. From Fig. 6(a), we can observe that both the computation time of FER-AM and *refiner* seems to quadratically increase as the problem size scales up, which might be due to the quadratically complexity of the attention mechanism [4] applied in AM and the accumulated attention mechanism with operations inside a $N \times N$ table in *refiner*. However, the increasing speed of computation time of the *refiner* is much slower than that of the whole FER-AM, which indicates low computational complexity of the designed *refiner*. From Fig. 6(b), we can observe that the computation time of FER-AM and the *refiner* seems to approximately linearly increase as the pool size increases, which is reasonable since the refiner needs to aggregate the information from all solutions in the pool. Moreover, the computation time of the refiner could be negligible compared to that of the whole model, which further demonstrates the computational efficiency of our method.

VI. Conclusions and Future Work

In this paper, we propose a novel encoder-refiner-decoder structure for solving vehicle routing problems, which iteratively improves a neural construction method by refining the feature embeddings from the encoder for broader search range and reconstructing a solution(s) with dynamic probability distributions accordingly via the decoder for more diverse search. To be specific, the proposed *refiner* first extracts graph topological features from dynamic historical solutions to derive the aggregated pool embeddings via an AGA layer, then improves the aggregated pool embeddings by absorbing the instructive and global information from previous improvement steps via a GRU cell, and finally refines the pre-trained



(a) Problem size (b) Pool capacity

Fig. 6: Computation time of the whole FER-AM and its *refiner* over different problem size and pool capacity.

feature embeddings from the encoder with the guidance of improved pool embeddings via a GNL layer. By doing so, the FER allows the neural construction methods to not only iteratively refine the feature embeddings for broader search range, but also dynamically update the probability distributions for more diverse search. Extensive experiments show that our method can effectively improve prevailing neural construction methods, and also exhibits competitive performance to the state-of-the-art neural improvement methods with much higher sampling efficiency. Given the genericalness and effectiveness of the proposed FER, it could be used in any encoder-decoder structured neural construction methods [10], [16], [21] to enhance their performance for solving routing problems. While our FER is effective, there are still some potential limitations: 1) its final performance may depend on the performance of the pre-trained backbone models; 2) its boost may be not significant for highly-optimized models (e.g., the state-of-the-art POMO) as discussed in Section V-C. In the future, we will investigate: 1) applying FER to ameliorate the out-of-distribution generalization performance; 2) considering effectively refining *improvement* methods; 3) applying FER to solve other combinatorial optimization problems like Bin Packing and Scheduling; 4) generalizing FER to larger problem sizes.

REFERENCES

[1] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European journal of operational research*, vol. 59, no. 3, pp. 345–358, 1992.

[2] P. Hansen, N. Mladenović, and J. A. M. Pérez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.

[3] K.-W. Pang, "An adaptive parallel route construction heuristic for the vehicle routing problem with time windows constraints," *Expert Systems with Applications*, vol. 38, no. 9, pp. 11939–11946, 2011.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[5] L. Xin, W. Song, Z. Cao, and J. Zhang, "Step-wise deep learning models for solving routing problems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4861–4871, 2020.

[6] L. Chen, B. Hu, Z.-H. Guan, L. Zhao, and X. Shen, "Multiagent meta-reinforcement learning for adaptive multipath routing optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5374–5386, 2021.

[7] Z. Zhang, H. Liu, M. Zhou, and J. Wang, "Solving dynamic traveling salesman problems with deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[8] Z. Zhang, Z. Wu, H. Zhang, and J. Wang, "Meta-learning-based deep reinforcement learning for multiobjective optimization problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[9] Z. Zhang, W. Jiang, J. Qin, L. Zhang, F. Li, M. Zhang, and S. Yan, "Jointly learning structured analysis discriminative dictionary and analysis multiclass classifier," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 8, pp. 3798–3814, 2017.

[10] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "POMO: Policy optimization with multiple optima for reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 21188–21198, 2020.

[11] R. A. Russell and W.-C. Chiang, "Scatter search for the vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 169, no. 2, pp. 606–622, 2006.

[12] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows," *Computers & operations research*, vol. 40, no. 1, pp. 475–489, 2013.

[13] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[14] M. Nazari, A. Oroojlooy, M. Takáč, and L. V. Snyder, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems*, pp. 9861–9871, 2018.

[15] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[16] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in *International Conference on Learning Representations*, 2018.

[17] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28, pp. 2692–2700, 2015.

[18] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *International Conference on Machine Learning (Workshop)*, 2017.

[19] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, pp. 6351–6361, 2017.

[20] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," tech. rep., arXiv preprint arXiv: 1906.01227, 2019.

[21] L. Xin, W. Song, Z. Cao, and J. Zhang, "Multi-decoder attention model with embedding glimpse for solving vehicle routing problems," in *AAAI Conference on Artificial Intelligence*, 2020.

[22] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang, "Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[23] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Advances in Neural Information Processing Systems*, vol. 32, pp. 6281–6292, 2019.

[24] A. Hottung and K. Tierney, "Neural large neighborhood search for the capacitated vehicle routing problem," in *24th European Conference on Artificial Intelligence*, 2020.

[25] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *International Conference on Learning Representations*, 2020.

[26] A. Hottung, B. Bhandari, and K. Tierney, "Learning a latent search space for routing problems using variational autoencoders," in *International Conference on Learning Representations*, 2021.

[27] Z. Ling, X. Tao, Y. Zhang, and X. Chen, "Solving optimization problems through fully convolutional networks: An application to the traveling salesman problem," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 12, pp. 7475–7485, 2020.

[28] M. Kim, J. Park, *et al.*, "Learning collaborative policies to solve np-hard routing problems," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[31] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.
[32] H. Taud and J. Mas, "Multilayer perceptron (mlp)," in *Geomatic approaches for modeling land change scenarios*, pp. 451–455, Springer, 2018.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[34] L. G. Optimization, "Gurobi optimizer reference manual," 2021.
[35] K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," *Roskilde: Roskilde University*, 2017.
[36] T. Vidal, "Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood," *Computers & Operations Research*, vol. 140, p. 105643, 2022.
[37] H. Awad, R. Elshaer, A. AbdElmo'ez, and G. Nawara, "An effective genetic algorithm for capacitated vehicle routing problem," in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, pp. 374–384, 2018.
[38] G. Reinelt, "TSPLIB-A traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.
[39] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017.

Jingwen Li received the B.E. degree in computer science from University of Electronic Science and Technology of China, China, in 2018, and the Ph.D. degree with the department of Industrial Systems Engineering and Management, National University of Singapore (NUS), in 2022. She is currently a lecturer in computer science of Sichuan Normal University. Her research interests include deep reinforcement learning and combinatorial optimization.

Yining Ma received the B.E. degree in computer science from the South China University of Technology, Guangzhou, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore. His research interests include learning to optimize, deep reinforcement learning, evolutionary computation, and combinatorial optimization.

Zhiguang Cao received the Ph.D. degree from Interdisciplinary Graduate School, Nanyang Technological University. He received the B.Eng. degree in Automation from Guangdong University of Technology, Guangzhou, China, and the M.Sc. in Signal Processing from Nanyang Technological University, Singapore, respectively. He was a Research Fellow with the Energy Research Institute @ NTU (ERI@N), a Research Assistant Professor with the Department of Industrial Systems Engineering and Management, National University of Singapore, and a Scientist with the Agency for Science Technology and Research (A*STAR), Singapore. He joins the School of Computing and Information Systems, Singapore Management University, as an Assistant Professor. His research interests focus on learning to optimize (L2Opt).

Yaoxin Wu received the B.Eng degree in traffic engineering from Wuyi University, Jiangmen, China, in 2015, the M.Eng degree in control engineering from Guangdong University of Technology, Guangzhou, China, in 2018, and the Ph.D. degree in computer science from Nanyang Technological University, Singapore, in 2023. He was a Research Associate with the Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU). He is currently an Assistant Professor with the Faculty of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology. His research interests include combinatorial optimization, integer programming and deep learning.

Wen Song received the B.S. degree in automation and the M.S. degree in control science and engineering from Shandong University, China, in 2011 and 2014, respectively, and the Ph.D. degree in computer science from Nanyang Technological University, Singapore, in 2018. He was a Research Fellow with the Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU). He is currently an Associate Professor with the Institute of Marine Science and Technology, Shandong University. His research interests include artificial intelligence, deep reinforcement learning, planning and scheduling, and operations research.

Jie Zhang received the Ph.D. degree from the Cheriton School of Computer Science, University of Waterloo, Canada, in 2009. He is currently a Full Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. He is also an Associate Professor at the Singapore Institute of Manufacturing Technology. During his Ph.D. study, he held the prestigious NSERC Alexander Graham Bell Canada Graduate Scholarship rewarded for top Ph.D. students across Canada. He was also a recipient of the Alumni Gold Medal at the 2009 Convocation Ceremony. The Gold Medal is awarded once a year to honour the top Ph.D. graduate from the University of Waterloo. His papers have been published by top journals and conferences and received several best paper awards. He is also active in serving research communities.

Yeow Meng Chee received the BMath, MMath, and Ph.D. degrees in computer science from the University of Waterloo in 1988, 1989, and 1996, respectively. He is currently Vice President (Innovation and Enterprise) at the National University of Singapore (NUS), Director of the NUS Enterprise Academy, as well as a Professor at the Department of Industrial Systems Engineering and Management of NUS. He was previously Vice Provost (Technology-Enhanced and Experiential Learning). Preceding his tenure at NUS, he was Head of the Division of Mathematical Sciences, Chair of the School of Physical and Mathematical Sciences, and Interim Dean of the College of Science at the Nanyang Technological University. His papers have been published by top journals and conferences. His research interests include coding theory, combinatorics, electronic design automation and theoretical computer science.

View publication stats