Xia Jiang[1], Yaoxin Wu[1], and Yingqian Zhang[1]

[1]Affiliation not available

April 08, 2024

Learning to Generate Hard Instances: Towards Robust Solutions for Vehicle Routing Problems

Xia Jiang, Yaoxin Wu, and Yingqian Zhang

***Abstract*—Deep models have shown promising results in solving vehicle routing problems (VRPs). However, existing models are often trained on instances from specific distributions and their worst-case performance is largely underexplored, thereby hindering the understanding and improvement of their robustness. In this paper, we present a generic framework to generate hard instances for obtaining robust VRP solutions. Given a pretrained deep model, we first develop an attack method that comprises an autoregressive sampling network (ASN) and a hardness measurement network (HMN). The two networks are trained alternately by reinforcement learning, aiming to generate hard instances for the given deep model and gauge the attack effect (i.e., hardness) of the instances, respectively. Then, we propose a simple yet effective training algorithm to robustify the deep model, which is progressively replaced by the continually trained HMN. Experimental results show that the attack method significantly degrades the performance of various deep models and conventional heuristics. Moreover, the training algorithm showcases the ability to enhance the robustness of the deep model, demonstrating its promising zero-shot generalizability.**

***Index Terms*—Vehicle routing problem, Neural network, Robustness, Deep reinforcement learning, Hard instance.**

I. Introduction

VEHICLE routing problems (VRPs), as one of the most studied branches in the realm of combinatorial optimization (CO), have a wide range of applications in logistics [1], manufacturing [2], etc. Nevertheless, VRPs pose a formidable challenge in terms of solving complexity owing to their NP-hard nature [3]. The pursuit of an optimal solution becomes notably intractable even for medium-sized problems [4], with the computational time rising dramatically as the problem size grows. While conventional heuristics are often manually crafted by much tuning work, the recent deep models draw upon the power of neural networks to automate the algorithmic design for solving VRPs. The learned heuristics have shown an advantageous balance between computational efficiency and solution quality [5], [6], [7], [8].

While achieving relatively good performance, current deep models for VRPs are generally vulnerable to out-of-distribution or out-of-size instances. Most of them suffer drastic performance degradation when solving instances that deviate from the specific distribution and size (of instances) used during the training process. Some approaches have been proposed to enhance the generalization of deep models in different distributions [9], [10], different sizes [11], [12], and both [13], [14]. In general, the above efforts simply train a deep model on instances with more than one random distribution or size, rendering it more generalizable to a broader range of VRP instances.

Despite the research on generalization, the worst-case performance of deep models is greatly underexplored. Such performance is critical to reveal the models' robustness when faced with extremely hard VRP instances. The stakeholders may estimate the reliability of deep models by observing their worst-case performance, and thereby decide if they can be applied in real-world scenarios. Intuitively, the worst-case performance of a model can be measured by solving hard-to-solve instances. To this end, identifying the distribution of the most hard instances is an effective way to understand and improve the robustness of deep models. A few existing studies exert relatively small perturbations on VRP instances to obtain adversarial, and hence hard, instances by adding a few nodes [15] or slightly modifying edges [4]. However, they are ineffective in finding more hard instances due to the perturbation limit, and lack the strategy to enhance deep models with the obtained instances.

In this paper, we propose a generic framework to generate model-specific distributions of hard instances for deep models. Moreover, the proposed framework can be employed to robustify a given model to achieve better performance on hard VRP instances. Given a pretrained deep model as the environment, we propose a deep reinforcement learning (DRL) method to attack the model. An autoregressive sampling network (ASN) is trained to sample hard sub-instances from the instances with larger problem sizes. During the attack, we also train a hardness measurement network (HMN) to gauge the attack effect (i.e., hardness of the generated sub-instances) and thus guide the training of ASN towards generating more hard instances. In addition, we propose a simple yet effective robustness-enhancing training (RET) algorithm to robustify the deep model. Similar to the attack, we initialize the HMN by the environment model and alternately train ASN and HMN. During training, the environment model is progressively replaced by the HMN. By doing so, the proposed framework keeps optimizing the environment model to deliver more robust solutions than the original model.

Our contributions are summarized as follows: 1) We propose a generic framework to discover hard instances for obtaining robust VRP solutions, which is applicable to different types of deep models. 2) We propose a DRL-based attack method to learn to generate model-specific distributions of hard instances for deep models. 3) The RET algorithm is developed based on the proposed framework to adaptively robustify the deep

The authors are with the Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands (email: summer142857.jiang@gmail.com; wyxacc@hotmail.com; yqzhang@tue.nl).

model, utilizing the generated hard samples. 4) We evaluate our framework on typical VRPs for various deep models and even conventional heuristics. The results show that the attack method is able to generate instances that significantly deteriorate the performance of deep models. Moreover, the RET can robustify deep models to perform better on hard instances, and meanwhile manifest promising zero-shot generalizability over unseen distributions and benchmark dataset.

II. Related work

A. Deep models for VRPs

Deep models utilize neural networks to approximate the optimal solution for VRPs. Existing deep models can be roughly categorized into two main paradigms [16]: learning-augmented models that adopt neural networks to enhance conventional heuristics, and end-to-end models that learn to construct solutions in sequential or one-shot manner. The former models integrate deep learning with existing domain knowledge of VRPs [17], such as improvement heuristics, large neighborhood search, and specialized solvers [18], [19], [20], [21], [22], [23], [24], [25], [26]. These models usually learn to refine an initial solution iteratively given specific contexts in the heuristics. In contrast, end-to-end models delve into constructing solutions from scratch via autoregressive neural networks trained by DRL [27], [28] or non-autoregressive neural networks trained by supervised learning [5], [29], [30]. End-to-end models often deliver comparable solutions to learning-augmented models with reduced inference time and are more researched in the literature. Some of the above models are further enhanced in terms of the generalization across different distributions or sizes [31], [6], [9], [10], [32], [11], [12], [13], [14]. However, they simply rely on additional training on instances with manually specified distributions (e.g., Uniform, Gaussian, Diagonal distributions) or sizes (e.g., random numbers of nodes within $[50, 200]$). The hard instances of deep models are still scarcely explored to better understand the robustness of deep models.

B. Robustness of deep models

The robustness of deep models can be measured by evaluations on the hard instances [4].

The authors of [15] maximize the cross-entropy over edges in VRP instances by adversarially inserting nodes (which follow the same distribution used in training). In [4], a DRL agent is trained to conduct problem-specific actions to generate adversarial instances of VRP solvers. The above works are restricted by slight modifications on clean instances, with the difficulty in estimating the hardness of more flexibly generated samples.

In contrast, a generative adversarial training (GAT) approach is proposed in [33], where the hardness of adversarial instances is measured by conventional solvers. However, the solver is prone to long-solving duration on large instances, instigating substantial training overhead. To tackle the issue, a hardness measurement approach is introduced in [34] to estimate the lower bound of the ground-truth optimality gap. On top of that, the authors perturb node coordinates of clean instances to generate instances with adaptive hardness. While achieving favorable attack effects, the gradient-dependent generation process is inapplicable to more general VRP solvers, e.g., conventional heuristics. In this paper, we propose a generic framework to generate hard instances for various deep models and even conventional heuristics. Meanwhile, our hardness measurement model can efficiently gauge attack effects of general hard instances, beyond adversarial instances generated by slight modifications.

III. Preliminaries

A. Vehicle routing problems

A VRP instance of problem size n is described over a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{v_i\}_{i=1}^{n}$ is the set of nodes (e.g., the depot or customers) and $\mathcal{E} = \{e_{i,j} | i, j \in [1, n], i \neq j\}$ represents the set of edges between any two different nodes v_i and v_j. A non-negative value $c_{i,j}$ is associated with each edge $e_{i,j} \in \mathcal{E}$ to represent the traveling cost from node v_i to v_j. A feasible solution (i.e., tour) is a sequence of nodes $\tau = (s_1, ..., s_k)$ with each element $s_i \in [1, n]$ being the index of a node in $\mathcal{V}$, i.e., $v_{s_i} \in \mathcal{V}$. Given a feasible solution, the total cost of the tour is obtained by $C(\tau|\mathcal{G}) = \sum_{i=1}^{n-1} c_{s_i, s_{i+1}}$. Assuming Ψ denotes the search space of feasible tours, the typical objective of VRPs is finding the optimal solution $\tau^*(\mathcal{G})$ with minimum total cost, as below,

$$\tau^*(\mathcal{G}) = \arg\min_{\tau \in \Psi} C(\tau|\mathcal{G}) \tag{1}$$

Different variants of VRP exist. For instance, the traveling salesman problem (TSP) aims to find a Hamiltonian circuit of the graph, which visits each node exactly once. In the capacitated vehicle routing problem (CVRP), a depot is specified and multiple sub-tours should be established, with all beginning and ending at the depot. Different from TSP, each node in CVRP is attributed not only by its coordinates but also the demand d_i. A feasible solution to a CVRP instance must visit each customer node exactly once and meanwhile keep total demand on each sub-tour below the vehicle capacity D. We consider Euclidean VRPs herein to specify $c_{i,j}$ as the Euclidean distance between any two nodes.

B. Hardness Metric

We regard hard VRP instances as the ones that significantly degrade a pretrained deep model. An effective attacker is supposed to consistently produce hard instances by non-trivial operations, rather than simple operations such as unlimited magnification of problem sizes. While obtaining an optimal solution $\tau^*(\mathcal{G})$ for a VRP instance $\mathcal{G}$ is intractable, a solution $\dot{\tau}(\mathcal{G}|\boldsymbol{\theta})$ derived by a well-trained deep model $f_{\boldsymbol{\theta}}$ can act as a fast approximate solution of $\tau^*(\mathcal{G})$. Formally, given an attacker G that produces Q hard instances $\{\tilde{\mathcal{G}}_i\}_{i=1}^{Q}$, their hardness for the deep model can be defined as below,

$$\mathcal{L}_r(\tilde{\mathcal{G}}, \tilde{\tau^*}|\boldsymbol{\theta}) = \frac{1}{Q}\sum_{i=1}^{Q} C(\dot{\tau}|\tilde{\mathcal{G}}_i, \boldsymbol{\theta}) - C(\tilde{\tau^*}|\tilde{\mathcal{G}}_i) \tag{2}$$

where $\tau^*(\tilde{\mathcal{G}})$ is shortened to $\tilde{\tau^*}$ for readability. The larger value of $\mathcal{L}_r(\tilde{\mathcal{G}}, \tilde{\tau^*}|\boldsymbol{\theta})$ stands for more significant attack effect, i.e.,

worse robustness of the deep model. The above hardness metric entails obtaining optimal solutions $\tilde{\tau}^*$, inevitably inducing heavy computational complexity. We propose the HMN in our framework to progressively estimate the above harness metric, which is delineated in subsubsection IV-A2.

Existing enhancements on the robustness of deep models are typically achieved via adversarial training [34], [33], which is commonly formulated as a min-max optimization problem,

$$\min_{\theta} \mathbb{E}_{(\mathcal{G},\tau^*)\sim\mu} \max_{\tilde{\mathcal{G}}\in\mathcal{N}_\omega(\mathcal{G})} \mathcal{L}_r(\tilde{\mathcal{G}}, \tilde{\tau^*}|\boldsymbol{\theta}) \tag{3}$$

where μ denotes the data distribution; $\mathcal{N}_\omega(\mathcal{G})$ denotes the attack space with attack budget ω, when $\mathcal{N}_\omega(\mathcal{G})$ usually defines a neighbourhood range of the original instance and ω defines the parameterized operating scale imposed to the instance, such as the numerical value of added noise [33] and gradient step size for updating [34]. Inspired by adversarial training, we propose the RET algorithm to robustify the deep model in subsection IV-B. It trains the attacker (i.e., ASN) to generate instances that maximize the hardness metric, and optimizes the deep model to minimize the cost of the generated instances in an alternate manner.

IV. Methodology

In this section, we first present the method to attack deep models for generating the associated distributions of hard instances. On top of that, we further extend the attack method to develop the RET algorithm, which adaptively robustifies the deep model with the generated hard instances.

A. Attack Method

As shown in Figure 1, the proposed framework consists of an ASN $G_{\boldsymbol{\beta}}$ (with parameters β) to generate hard instances, and an HMN $H_{\boldsymbol{\phi}}$ (with parameters ϕ) to evaluate the hardness of the generated instances. Given a pretrained deep model as the environment, we train the ASN by DRL algorithm to sample hard sub-instances $\{\tilde{\mathcal{G}}_i\}_{i=1}^{Q}$ from random instances $\{\mathcal{G}_i\}_{i=1}^{Q}$, in order to degrade the deep model performance as much as possible. We set the downsampling factor $\omega(\omega < 1)$ to represent that the size of original random instances n_o is $1/\omega$ times the size of the sampled hard sub-instances n_a, i.e., $n_o = \omega n_a$. By doing so, different distributions of hard instances under any specified problem size can be generated by adjusting ω and n_o. Notably, we treat the deep model as a black-box environment in the training of ASN, without necessary access to its neural structure, gradients, etc. It implies the generality of our attack method to be applied with broad deep models.

Ideally, the ASN can be optimized by directly maximizing the hardness metric in Equation 2. However, to bypass the heavy computation of optimal solutions, we train the HMN $H_{\boldsymbol{\phi}}$ on the generated instances to estimate the optimal solutions with their objective values. Given the objective values derived from $H_{\boldsymbol{\phi}}$ and the environment model, we readily gain the approximate hardness of the generated instances, which is employed as the reward signal to guide the training of ASN.

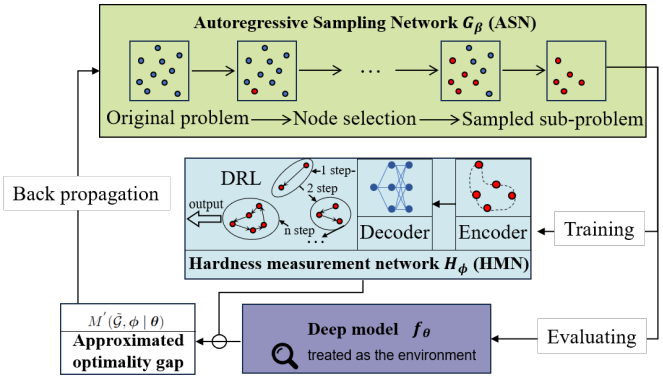

Fig. 1: The illustration of the proposed framework.

1) Autoregressive sampling network: Inspired by [27], we also structure the proposed ASN with the attention-based encoder-decoder architecture.

Encoder. Given an instance $\mathcal{G}$, its node features $\mathbf{v}^o \in \mathbb{R}^{n_o \times d_o}$ are taken as input, where d_o is the feature dimension (e.g., $d_o = 2$ for TSP and $d_o = 3$ for CVRP), the encoder comprises a d_h-dimensional embedding layer and N attention blocks. In the embedding layer with parameters $\mathbf{W}^e \in \mathbb{R}^{d_o \times d_h}$ and $\mathbf{b}^e \in \mathbb{R}^{d_h}$, the input $\mathbf{v}^o$ is linearly transformed by $\mathbf{h}^{(0)} = \mathbf{W}^e\mathbf{v}^o + \mathbf{b}^e$. Subsequently, $\mathbf{h}^{(0)}$ is processed by attention blocks, each of which comprises a multi-head attention (MHA) layer [35], a node-wise feed-forward (FF) layer, along with skip-connection [36] and batch normalization (BN) [37]:

$$\hat{\mathbf{h}}^{(l)} = \mathbf{BN}_l(\mathbf{h}^{(l-1)} + \mathbf{MHA}_l(\{\mathbf{h}_1^{(l-1)}, ..., \mathbf{h}_{n_o}^{(l-1)}\})) \tag{4}$$

$$\mathbf{h}^{(l)} = \mathbf{BN}_l(\hat{\mathbf{h}}^{(l)} + \mathbf{FF}_l(\hat{\mathbf{h}}^{(l)})) \tag{5}$$

where $l \in [1, N]$ denotes the block index. **MHA** and **FF** are designed to keep d_h dimensions of $\mathbf{h}^{(l)}$, i.e., $\mathbf{h}^{(l)} \in \mathbb{R}^{n_o \times d_h}$.

The **MHA** mechanism empowers the deep model to attend to different parts of the input sequence, and parallelly deliver more advanced representations of the parts. In this paper, we employ **MHA** with $M = 8$ heads to execute the attention operation and update node embeddings in the encoder. For head m of block l in the encoder, node embeddings $\mathbf{h}^{(l-1)} \in \mathbb{R}^{n_o \times d_h}$ are taken as input and transformed to obtain query embeddings $\mathbf{Q}_m^l$, key embeddings $\mathbf{K}_m^l$ and value embeddings $\mathbf{V}_m^l$, as below,

$$\mathbf{Q}_m^l = \mathbf{W}_m^q\mathbf{h}^{(l-1)},\ \mathbf{K}_m^l = \mathbf{W}_m^k\mathbf{h}^{(l-1)},\ \mathbf{V}_m^l = \mathbf{W}_m^v\mathbf{h}^{(l-1)} \tag{6}$$

where $\mathbf{W}_m^q, \mathbf{W}_m^k, \mathbf{W}_m^v \in \mathbb{R}^{d_h \times d_A}$ ($d_A = 16$) are learnable matrices. Subsequently, the query, key, value embeddings are used to compute the d_A-dimensional node embeddings in each head m ($\forall m \in \{1, \ldots, M\}$), such that,

$$a_m(\mathbf{Q}_m^l, \mathbf{K}_m^l, \mathbf{V}_m^l) = \text{Softmax}(\frac{\mathbf{Q}_m^l\mathbf{K}_m^l}{\sqrt{d_A}})\mathbf{V}_m^l \tag{7}$$

after which the outputs from all heads are concatenated and transformed by another weight matrix $W_c^l \in \mathbb{R}^{d_h \times d_h}$ to obtain the advanced d_h-dimensional node embeddings below,

$$\mathbf{MHA}_l(\mathbf{Q}_m^l, \mathbf{K}_m^l, \mathbf{V}_m^l) = \text{concat}(a_1, a_2, ..., a_M)W_c^l \tag{8}$$

The **FF** layer computes node-wise projections by the affine transformations with a ReLu activation function [38]. Given node embeddings $\hat{\mathbf{h}}^{(l)}$ as input, the **FF** layer in block l processes them by the following equation,

$$\mathbf{FF}_l(\hat{\mathbf{h}}^{(l)}) = \mathbf{W}^l_{\text{ff},1} \cdot \text{ReLu}(\mathbf{W}^l_{\text{ff},0}\hat{\mathbf{h}}^{(l)} + \mathbf{b}^l_{\text{ff},0}) + \mathbf{b}^l_{\text{ff},1} \quad (9)$$

where $\mathbf{W}^l_{\text{ff},0} \in \mathbb{R}^{d_h \times d_f}$, $\mathbf{W}^l_{\text{ff},1} \in \mathbb{R}^{d_f \times d_h}$, $\mathbf{b}^l_{\text{ff},0} \in \mathbb{R}^{d_f}$, $\mathbf{b}^l_{\text{ff},1} \in \mathbb{R}^{d_h}$ are learnable parameters, and we set $d_f = 512$.

Decoder. The decoder takes as input the node embeddings $\mathbf{h}^{(N)}$ derived from the encoder, and sequentially samples nodes to form sub-instances. While current end-to-end deep models [28], [39] often sample nodes to construct feasible VRP solutions, We decode the nodes in a VRP instance to discover a hard sub-instance with n_a nodes. Since the most intensive computation of the autoregressive model is led by long sequence generation in the decoding, the ASN with a fixed n_a does not significantly increase computational memory and time when n_o grows. In that regard, we can generate instances with different levels of hardness by adjusting n_o, without introducing extra training overhead.

Specifically, the embedding $\hat{\mathbf{h}}^i_{(c)} = [\mathbf{h}^i_{t-1}, \mathbf{h}^i_1]$ is extracted in each decoding step $t \in \{2, ..., n_a\}$ as a reflection of the decoding context, where $\mathbf{h}^i_{t-1}$, $\mathbf{h}^i_1 \in \mathbf{h}^{(N)}$ are embeddings of nodes sampled at step $t-1$ and step 1, respectively. Specially, $\hat{\mathbf{h}}^i_{(c)} = [\mathbf{h}^i_1]$ at step 1. To facilitate the exploration of hard instances, we enable the parallel decoding to generate n_a sub-instances concurrently. It means that n_a nodes are randomly selected from the original instance $\mathcal{G}$, with their embeddings extracted at step 1 to be different contexts. We adopt the index $i \in \{1, ..., n_a\}$ to signify the i-th step in the parallel decoding.

In each decoding step t, the context embedding $\hat{\mathbf{h}}^i_{(c)}$ aggregates all node embeddings through MHA, such that,

$$\mathbf{Q}^i = \mathbf{W}^q\hat{\mathbf{h}}^i_{(c)}, \quad \mathbf{K}^i = \mathbf{W}^k\mathbf{h}^{(N)}, \quad \mathbf{V}^i = \mathbf{W}^v\mathbf{h}^{(N)} \quad (10)$$

$$\mathbf{h}^i_{(c)} = \mathbf{MHA}(\mathbf{Q}^i, \mathbf{K}^i, \mathbf{V}^i) \quad (11)$$

where $\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v \in \mathbb{R}^{d_h \times d_A}$ are all learnable parameters, with d_A being the dimension of attention parameters. By doing so, the updated context embedding $\mathbf{h}^i_{(c)}$ involves the information of the original instance $\mathcal{G}$ and the sub-instance at current step t. Subsequently, we compute the attention logit of selecting node j to be added in the sub-instance i, as below,

$$\text{logit}^i_j = \begin{cases} C \cdot \tanh(\frac{\mathbf{h}^i_{(c)}\mathbf{h}^i_j}{\sqrt{d_A}}) & \text{if } j \neq \mathbf{v}_t, \forall i \in \{1, ..., n_a\}, \\ -\infty & \text{otherwise} \end{cases} \quad (12)$$

where $\mathbf{v}_t$ represents the set of nodes that are sampled before t. Following the literature [27], we clip logits into $[-C, C](C = 10)$ for calculating appropriate probabilities of choosing each node:

$$p^i_{(\boldsymbol{\beta})}(v_t = j|\mathcal{G}, \mathbf{v}_t) = e^{\text{logit}^i_j} / \sum_{k=0}^{n_a} e^{\text{logit}^i_k} \quad (13)$$

with which we sample the next node and add it to the sub-instance i. Among n_a generated sub-instances from the original instance, we gauge their harnesses by the HMN and determine the hardest one as the final generated instance.

2) Hardness measurement network: As aforementioned, the cost difference $C(\dot{\tau}|\tilde{\mathcal{G}}, \boldsymbol{\theta}) - C(\tilde{\tau^*}|\tilde{\mathcal{G}})$ in Equation 2 or the relative difference (i.e., optimality gap) $M(\tilde{\mathcal{G}}) = (C(\dot{\tau}|\tilde{\mathcal{G}}, \boldsymbol{\theta}) - C(\tilde{\tau^*}|\tilde{\mathcal{G}}))/C(\tilde{\tau^*}|\tilde{\mathcal{G}})$ could be an ideal metric to reflect the hardness of $\tilde{\mathcal{G}}$, since it measures the performance of deep model $f_{\boldsymbol{\theta}}$ in comparison to the optimal solution $\tilde{\tau^*}$. Given the intractability to acquire $\tilde{\tau^*}$ in practice, we develop the HMN H_ϕ as a surrogate model to obtain a (near-)optimal solution $\dot{\tau}(\tilde{\mathcal{G}}|\boldsymbol{\phi})$.[1] Consequently, we replace $\tilde{\tau^*}$ with $\dot{\tau}(\tilde{\mathcal{G}}|\boldsymbol{\phi})$ and formulate an approximate optimality gap $M^{'}(\tilde{\mathcal{G}})$, i.e., a lower bound of the ground-truth gap, such that,

$$M^{'}(\tilde{\mathcal{G}}, \boldsymbol{\phi} \mid \boldsymbol{\theta}) = \frac{C(\dot{\tau}|\tilde{\mathcal{G}}, \boldsymbol{\theta}) - C(\dot{\tau}|\tilde{\mathcal{G}}, \boldsymbol{\phi})}{C(\dot{\tau}|\tilde{\mathcal{G}}, \boldsymbol{\phi})} \leq M(\tilde{\mathcal{G}}) \quad (14)$$

where the equality is satisfied if the HMN attains the optimal solution. Intuitively, the HMN should be trained to gain near-optimal solutions for the generated hard instances. To this end, we employ the pretrained POMO (Policy Optimization with Multiple Optima) model as the HMN [28]. [2], and continue training it on the hard instances. While other deep models for VRPs could instantiate HMN, our experiments manifest the promising attack effect with POMO.

3) Alternate training scheme: While the ASN is trained to generate hard instances, the HMN is trained to solve the instances and gain near-optimal solutions. Then, the approximate optimality gap in Equation 14 is attained as the reward in the DRL training of ASN, thereby guiding it to generate harder instances. With the above, we alternatively train ASN and HMN to improve their performance progressively. More specifically, we train the two neural networks by REINFORCE algorithm [40]. Given a batch of $\mathcal{B}$ random instances, ASN generates $n_a\mathcal{B}$ hard instances in parallel. Subsequently, the HMN is updated by the following gradient ascent,

$$\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) \approx -\frac{1}{n_a\mathcal{B}} \sum_{i=1}^{n_a\mathcal{B}} (C(\boldsymbol{\tau}^i) - b^H_i(\tilde{\mathcal{G}}_i)) \nabla_{\boldsymbol{\phi}} \log p_{(\boldsymbol{\phi})}(\boldsymbol{\tau}^i \mid \tilde{\mathcal{G}}_i) \quad (15)$$

Following autoregressive deep models, we obtain $p_{(\boldsymbol{\phi})}(\boldsymbol{\tau}^i \mid \tilde{\mathcal{G}}_i) = \prod_{t=2}^{n_a} p_{(\boldsymbol{\phi})}(v^i_t|\tilde{\mathcal{G}}_i, \mathbf{v}^i_{1:t-1})$ by POMO. We adopt a baseline function $b^H_i(\tilde{\mathcal{G}}_i)$ to estimate the quality of a solution in comparison to the average cost, which is defined as $b^H_i(\tilde{\mathcal{G}}_i) = \frac{1}{n_a}\sum_{j=1}^{n_a} C(\boldsymbol{\tau}^i), \forall i \in [1, n_a\mathcal{B}]$.

With the updated HMN, we could gain a better approximation of the optimality gap, which reflects the hardness of the generated instances. For maximizing the hardness, we apply the approximate gap as the reward to update ASN by the following gradient ascent,

$$\nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) \approx -\frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} (M^{'}(\tilde{\mathcal{G}}_i, \boldsymbol{\phi} \mid \boldsymbol{\theta}) - b^A_i(\tilde{\mathcal{G}}_i)) \nabla_{\boldsymbol{\beta}} \log p_{(\boldsymbol{\beta})}(\tilde{\mathcal{G}}_i \mid \mathcal{G}_i) \quad (16)$$

Similarly, we define $b^A_i(\tilde{\mathcal{G}}_i) = \frac{1}{n_a}\sum_{j=1}^{n_a} M^{'}(\tilde{\mathcal{G}}_i, \boldsymbol{\phi} \mid \boldsymbol{\theta}), \forall i \in [1, \mathcal{B}]$ to reflect the average hardness of n_a parallelly generated instances. As multiple sub-problems are generated by the ASN from a single larger-size instance, they produce a good baseline for estimating the quality of a specific hard instance. The utilized advantage function $b^A_i(\tilde{\mathcal{G}}_i)$ can thus reduce gradient variance and thus increase the learning speed.

[1] Interestingly, we demonstrate in subsection V-E that using a relatively weak model as the HMN gains better attack performance than a strong solver to some extent.

[2] https://github.com/yd-kwon/POMO/tree/master

B. Robustness-enhancing Training

Algorithm 1 Robustness-enhancing training algorithm

Input: Environment model $f_{\boldsymbol{\theta}}$; random-initialized ASN $G_{\boldsymbol{\beta}}$; the HMN instantiated by $H_{\boldsymbol{\phi}} \leftarrow f_{\boldsymbol{\theta}}$
Output: Robustness-enhanced model $f^*_{\boldsymbol{\theta}}$
Parameter: Number of rounds, epochs T_r, T_p; problem sizes n_o, n_a; learning rates $\eta_{\boldsymbol{\beta}}$, $\eta_{\boldsymbol{\phi}}$

```
Initial round t_r ← 0 and Initial epoch t_p ← 0
for t_r ← 0 to T_r do
    Randomly Initialize β.
    for t_p ← 0 to T_p do
        for a batch of instances {G_i}_{i=1}^{B} with size n_o do
            Sample hard instances using G_β with n_a multiple starting nodes.
            Update H_φ by φ ← φ − η_φ ∇_φ J(φ).
            Update G_β by β ← β − η_β ∇_β J(β).
        end for
    end for
    Re-initialize the environment model by θ ← φ
end for
return f*_θ = f_θ
```

The RET algorithm can be developed naturally when the HMN and environment models share the same neural structure. In this case, we employ the same pretrained deep model (e.g., POMO) to serve as the HMN and environment model before training ASN. As mentioned above, the HMN is continuously trained by hard instances generated by ASN, implying that the min-max optimization in Equation 3 is carried out spontaneously. To further robustify the environment model, we stipulate that the model is replaced by the HMN for every T_p epoch. More specifically, after one round (i.e., T_p epochs) of alternate training with ASN and HMN, the parameters of the environment model are updated by those in HMN. In the next round, the ASN is randomly reinitialized to learn how to generate hard instances for the new environment model (i.e., robustified HMN from the last round). After rounds of training and updates, the robustness of the original deep model is expected to be enhanced. The workflow of the RET algorithm is presented in Algorithm 1. We will show in experiments that the zero-shot generalizability of the deep model can also be improved by the RET, without additional training efforts.

V. Experiments

We conduct experiments on two typical VRP tasks, i.e., TSP and CVRP. Node coordinates are randomly chosen from the unit square according to the literature [27], [28]. Various deep models for VRPs are attacked, indicating the generality of the proposed framework. Specifically, we learn hard instances to attack POMO [28], Simulation-guided Beam Search (SGBS) [41], Adaptive Multi-distribution Knowledge Distillation (AMDKD) [10], and Omni-VRP [13], which serve as the environment models, respectively. These pretrained models are available at their repositories.[3][4][5] Meanwhile, we utilize LKH3[6] to calculate the (near-)optimal solution for both TSP and CVRP, to obtain optimality gaps. We perform 10 runs using LKH for all the problems. All experiments are carried out on a machine with an AMD EPYC 7F72 CPU at 3.2 GHz and an NVIDIA A100 40G GPU.

A. Training Setups

The hyperparameters of the environment models are all consistently derived from the original paper. As we use POMO as the HMN, it follows most of the setups in [28]. We use Adam optimizer [42] to fine-tune the HMN, and the corresponding learning rate $\eta_{\boldsymbol{\phi}}$ is set to $1e-4$. We use $\times 8$ augmentation to enhance HMN for better evaluating the hardness of generated instances. The neural structure of ASN is the same as POMO, which has $N = 6$ attention blocks in the encoder. We also utilize Adam optimizer to train the ASN, with a learning rate $\eta_{\boldsymbol{\beta}} = 5e-5$. 1000 instances $\{\mathcal{G}_i\}_{i=1}^{1000}$ are used in each epoch of training with a batch size of 64, and we only use 5 epochs (i.e., 5000 random instances) to train the randomly initialized ASN. Compared to existing deep models, which usually learn from millions of instances, the training of ASN is much more efficient.

We train the ASN to generate hard instances with two sizes, i.e., $n_a = 50$ and 100, respectively. However, we only use the pretrained models trained on instances of size 100 as the environments (except Omni-VRP, which has been trained on different-sized problems). By these settings, we evaluate the performance on seen and unseen problem sizes (i.e., 100 and 50) in the training of deep models. For each setting, we train the ASN with the downsampling factor $\omega = 1/2$, $1/4$, and $1/6$, respectively. Given the fixed size of generated instances, different values of ω help understand the influence of enlarging the size n_o of random instances on the generated instances. In addition, $\times 8$ augmentation is incorporated in the environment model for all training and evaluation cases.

B. Baselines

The baselines involved to show the attack effect are: 1) Clean instances: They are uniformly generated samples, to show the performance of deep models before being attacked. 2) Random sampling: We generate larger instances following uniform distribution, and randomly sample sub-instances from them with $\omega = 1/6$, to demonstrate the straightforward and random attack. 3) Hardness-adaptive Model (HAM) [34]: The state-of-the-art framework for both attacking and defending deep models of VRPs. We apply our HMN as the hardness evaluator in the attack model of HAM for fair comparison. We also compare the defense method in HAM, i.e., hardness-adaptive curriculum, with our RET algorithm.

In addition to the above baselines, we further compare our framework with attack methods specialized for specific VRPs, including: 1) Perturbation [15], which attacks the Graph

[3]https://github.com/yd-kwon/SGBS/tree/main
[4]https://github.com/jieyibi/AMDKD/tree/main
[5]https://github.com/RoyalSkye/Omni-VRP/tree/main
[6]http://webhotel4.ruc.dk/ keld/research/LKH-3/

	Solver	Problem size	Optimality gap (%)					
			Clean instances	Random sampling	HAM	ASN ($\omega = 1/2$)	ASN ($\omega = 1/4$)	ASN ($\omega = 1/6$)
TSP	POMO	50	0.07 (0.21)	0.06 (0.21)	6.28 (5.69)	4.32 (1.88)	5.96 (2.02)	**9.15 (2.42)**
	SGBS	50	0.02 (0.14)	0.02 (0.10)	-	2.02 (1.09)	2.72 (1.28)	**4.10 (1.50)**
	AMDKD	50	0.14 (0.27)	0.14 (0.27)	0.32 (0.45)	0.98 (0.58)	**1.44 (0.77)**	1.15 (0.54)
	Omni-VRP	50	0.87 (0.98)	0.90 (1.04)	3.77 (2.12)	6.07 (1.74)	4.45 (1.38)	**9.84 (2.29)**
	POMO	100	0.13 (0.23)	0.15 (0.25)	32.85 (11.06)	6.84 (1.75)	22.30 (3.97)	**46.98 (4.96)**
	SGBS	100	0.06 (0.15)	0.06 (0.17)	-	2.28 (1.41)	22.78 (4.35)	**29.80 (3.52)**
	AMDKD	100	0.35 (0.35)	0.35 (0.35)	0.88 (0.65)	1.71 (0.63)	2.60 (0.76)	**3.18 (0.82)**
	Omni-VRP	100	1.27 (0.81)	1.30 (0.85)	5.38 (2.03)	6.53 (1.18)	9.07 (1.25)	**13.31 (2.05)**
CVRP	POMO	50	3.20 (1.55)	3.07 (1.49)	13.69 (4.51)	11.92 (3.24)	19.07 (3.78)	**23.51 (4.52)**
	SGBS	50	1.49 (1.08)	1.35 (1.10)	-	7.60 (2.51)	**14.03 (3.50)**	13.38 (5.10)
	AMDKD	50	6.23 (2.42)	6.15 (2.41)	**38.78 (9.09)**	19.35 (4.28)	28.81 (4.56)	30.78 (5.56)
	Omni-VRP	50	3.30 (1.45)	3.30 (1.43)	3.42 (1.72)	12.61 (2.75)	17.33 (3.50)	**18.90 (3.61)**
	POMO	100	1.92 (0.88)	1.90 (0.94)	4.24 (2.64)	6.43 (1.48)	10.67 (1.97)	**13.33 (3.01)**
	SGBS	100	1.28 (0.81)	1.26 (0.93)	-	4.44 (1.43)	6.80 (2.01)	**8.61 (2.74)**
	AMDKD	100	2.26 (0.94)	2.27 (0.95)	2.10 (1.25)	5.66 (1.41)	9.50 (1.85)	**10.15 (2.11)**
	Omni-VRP	100	3.36 (1.12)	3.35 (1.10)	2.90 (1.40)	9.27 (1.57)	**15.17 (2.70)**	11.19 (4.89)

TABLE I: Attack performance on TSP and CVRP. The (near-)optimality gaps are calculated by LKH-3.0.8.

Neural Network (GNN) model [5] on TSP task. This attack method adversarially inserts 5 nodes to a clean instance, making the solver perform as poorly as possible on new instance. 2) ROCO (Robust Combnaotorial Optimization) [4], which attacks the MatNet model [43] on the asymmetric traveling salesman problem (ATSP) task. This method applies a proximal policy optimization algorithm and trains an attacker agent, which is able to adversarially modify edges in the graph of an instance. For fair comparison, we implement the ASN in our framework with the same deep models as used in Perturbation and ROCO methods, i.e., GNN and MatNet, respectively.

C. Attack Performance

We first evaluate our attack method and baselines on different deep models. Given each deep (environment) model to attack, we generate 1000 instances by the trained ASN and baselines, respectively. Then, we test the deep model on the instances generated by each attack method. We display the average optimality gaps for TSP and CVRP in Table I, respectively, where the standard deviations of gaps are in the brackets. The results manifest the effectiveness of attack methods, i.e., how much they degrade the tested deep models, and meanwhile reflect their robustness to hard instances.

We observe that random sampling performs on par with using clean instances. It indicates that the significant attack effect by our method does not come from simple sampling but is due to the effective ASN that generates more hostile instances. Generally, decreasing ω makes the generated instances (of the same size) harder. Taking the case of POMO on TSP100 as an example, the ASN with $\omega = 1/2$, $1/4$ and $1/6$, increases the optimality gap of POMO from 0.13% (on clean instances) to 6.84%, 22.30%, and 46.98% respectively. The devastating effect of deteriorating the performance of POMO by around 361 times with $\omega = 1/6$ clearly indicates the need of improving its robustness. Comparing to HAM, while the HAM can significantly degrade AMDKD on CVRP with size 50, our attack method with $\omega = 1/6$ is more effective in other scenarios. The standard deviation of gaps generated by HAM is relatively larger than ASN, suggesting that our method produces hard instances more steadily.

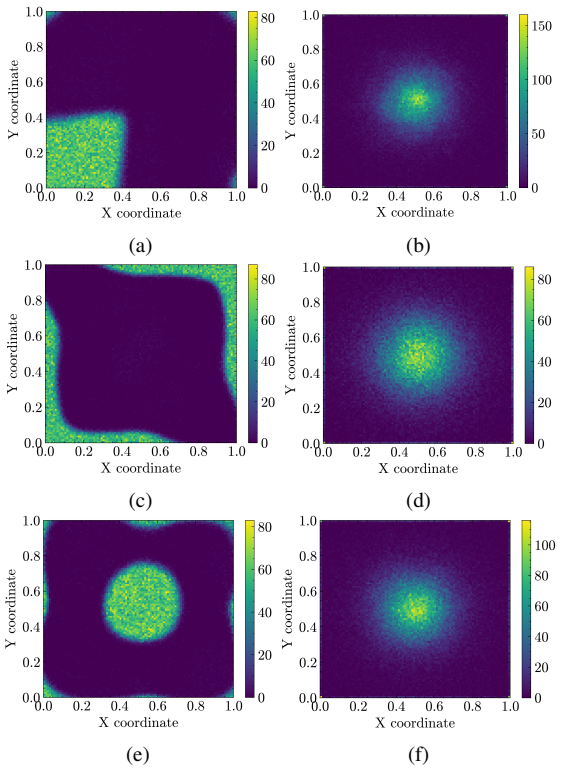

Fig. 2: Distributions of hard TSP instances (of size 100) generated by ASN and HAM for three deep models: (a) POMO (ASN), (b) POMO (HAM), (c) AMDKD (ASN), (d) AMDKD (HAM), (e) Omni-VRP (ASN), (f) Omni-VRP (HAM).

1) Visualization of hard instances: To observe the distributions of hard instances, we visualize 1000 generated instances for three deep models by two-dimensional histograms. We

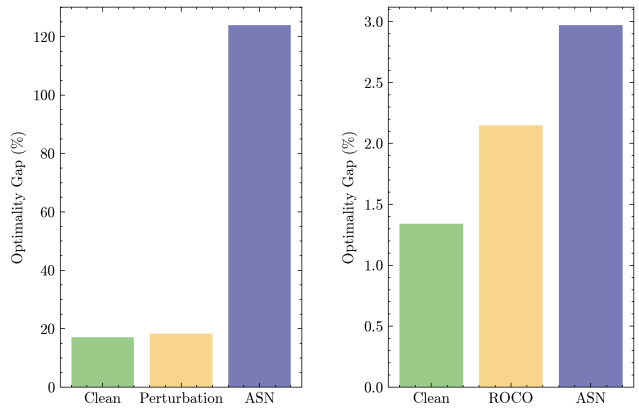

Fig. 3: Comparison of ASN and other attack methods on (a) GNN (TSP) and (b) MatNet (ATSP).

set 100 bins in each figure and depict the frequency of nodes in the generated instances, as shown in Figure 2. While all the distributions of hard instances deviate from Uniform distribution, the instances generated by HAM tend to consistently form a cluster, showing monotonous patterns for different deep models. Instead, our ASN produces model-specific distributions, thereby delivering a better attack effect.

2) Robustness of deep models: By comparing the attack performance on deep models, we find the disparity in their robustness. For example, POMO is more vulnerable to attack, given the fact that it was consistently trained on Uniform distribution. AMDKD, which was trained across distributions, shows generally better robustness and is not prone to be attacked, particularly on the TSP task. However, as we use the AMDKD model trained on problems of size 100, its performance drops dramatically on CVRP instances of size 50, showing its weakness in keeping robust when the size of test cases is different from its training data.

3) Comparison study with other attack models: We also implement the proposed framework for attacking the GNN model on TSP (with 100 nodes) and the MatNet model on ATSP (with 50 nodes). For the GNN model, we re-train the model based on the original dataset in [5] and use greedy search for decoding. For the MatNet model [43], we use its encoder as the encoder of our ASN to cope with matrix input. The comparative results are displayed in Figure 3. As shown, our framework manifests superior attack performance, and delivers much larger optimality gaps than Perturbation and ROCO methods in [15], [4]. Notably, these two attack methods are only applicable to GNN and MatNet models for TSP and ATSP, respectively. In contrast, our framework can be applied to attack any type of deep models (i.e., POMO, GNN, and MatNet) for different VRPs (i.e., TSP, CVRP, ATSP). In summary, our framework in comparison with existing attack methods is more versatile to attack various deep models, and can gain better attack performance on different VRPs.

4) Attack on conventional heuristics: As our method does not acquire any prior knowledge of environment models, we also perform experiments on conventional heuristics, including nearest neighbour, nearest insertion, and farthest insertion [44], [45]. Given the low efficiency of these methods in solving relatively large VRPs, we only take TSP with size 50 as an example w.r.t $\omega = 1/4$. The results in Table II reveal that the ASN can attack the heuristics significantly to degrade all heuristics by much larger optimality gaps.

Heuristic	Clean instances	Hard instances
Nearest Neighbour	3.54% (2.50%)	9.62% (2.13%)
Nearest Insertion	6.09% (3.82%)	53.39% (7.98%)
Farthest Insertion	4.79%(2.92%)	33.75% (8.28%)

TABLE II: Attack performance on TSP with size 50. Average optimality gaps of heuristics for clean and hard instances are calculated.

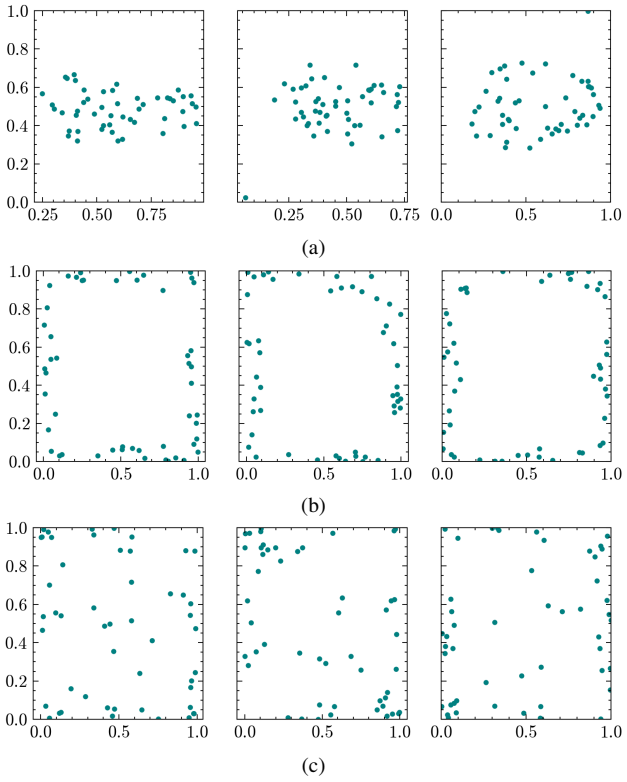

Fig. 4: The hard-to-solve examples of heuristic solvers. (a) Nearest Neighbour, (b) Nearest Insertion, (c) Farthest Insertion.

We present some hard instances for conventional construction heuristics in Figure 4. It is obvious that nodes in hard instances are distributed differently w.r.t different construction heuristics. Specifically, the nodes exhibit a change from coalesce to clusters when subjected to the nearest neighbor algorithm, whereas the nodes are dispersed along the peripheries in hard instances of the nearest insertion algorithm. It indicates that our attack method can identify distributions of hard instances for generic VRP algorithms, offering valuable insights into the their vulnerabilities in worst-case scenarios.

D. RET Performance

To verify the effectiveness of the RET algorithm, we employ it to robustify POMO as an example. We set $T_r = 20$ and $T_p = 5$, and perform experiments on TSP and CVRP of size

	Method	Average optimality gap		
		Robustness	Generalization	Benchmarking
TSP	POMO	39.92%	3.40%	3.07%
	POMO_H	8.04%	4.91%	5.30%
	POMO_A	8.39%	4.78%	4.80%
	POMO_G	15.32%	2.11%	2.58%
	POMO_D	29.75%	2.02%	3.04%
	POMO_R	4.50%	1.13%	2.15%
CVRP	POMO	8.79%	2.00%	8.59%
	POMO_H	6.51%	2.04%	8.88%
	POMO_A	6.67%	2.04%	8.01%
	POMO_G	12.62%	1.74%	8.17%
	POMO_D	12.04%	2.33%	9.94%
	POMO_R	7.10%	1.89%	7.54%

TABLE III: Comprehensive performance of different POMO models.

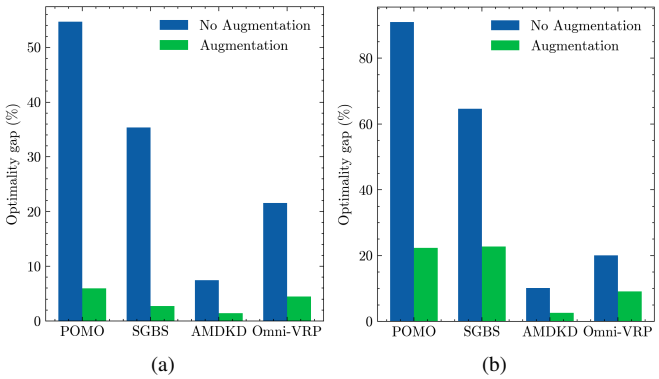

Fig. 5: Performance of deep models in solving hard TSP instances of size (a) 50 and (b) 100, with and without instance augmentation.

100. To ensure that the advantage of RET is not from simple training on more data, we continue fine-tuning the pretrained POMO on 10M instances from HAM, Gaussian mixture distribution (POMO_G) and Diagonal distribution (POMO_D) for fair comparison. For the instance generated by HAM, we use the hardness-adaptive curriculum learning (POMO_H), which is proposed in [34], and the vanilla adversarial training (POMO_A), which is defined by Equation 3, to further train the model, respectively. The models are evaluated in three aspects, including robustness (by solving hard instances from attack methods), zero-shot generalizability (by solving instances from unseen distributions), and benchmarking performance (by solving real-world instances).

1) Robustness Enhancement: We evaluate robustness of the RET-trained POMO (i.e., POMO_R) and baselines by attacking them with both HAM and ASN ($\omega = 1/6$). In specific, they are adopted to generate 1000 hard instances for RET-trained POMO and each baseline, respectively. We test all POMO models on respective hard instances to see whether their robustness is enhanced compared to the original pretrained POMO. We report in Table III the average optimality gap over all hard instances for each model. As shown, the gap is largely reduced after POMO is defended by RET. However, the effect of fine-tuning on random distributions generally attain ambiguous performance. Therefore, the RET is a promising method to enhance the robustness of deep models, with the potential to improve the worst-case performance.

2) Zero-shot Generalizability: To assess zero-shot generalizability, we test the RET-trained POMO and baselines on the dataset which incorporates different distributions. Besides Uniform distribution, we employ commonly used distributions in the literature, including Gaussian mixture distribution [13], Cluster distribution [7], Diagonal distribution [33], and Explosion distribution [46]. We produce 1000 instances per distribution for the test. Notably, distributions (except Uniform distribution) are unseen in the RET process, and thus the performance on them reflects the zero-shot generalizability. Details of the distributions are provided in Appendix A.

The average optimality gap over all instances is recorded for each model in Table III, and we present detailed results on each distribution in Appendix A. As revealed, although we did not explicitly include aforementioned distributions in the RET process, the robustified POMO significantly outperforms the original POMO on average. It indicates that the RET can effectively enhance the zero-shot generalizability. In comparison to POMO models fine-tuned on specific distributions, i.e., POMO_G and POMO_D, the RET-trained POMO generally attains superior zero-shot generalizability, without exposure to any distribution (except Uniform distribution).

3) Benchmarking: We test our robustified POMO and baselines on the benchmark datasets, including TSPLIB [47] and CVRPLIB (Set-X) [48], which are extracted from real-world routing scenarios. We solve a set of representative instances with problem size within $[50, 300]$, and record the average performance in Table III. We observe that the RET-trained POMO outperforms all baselines and gain the smallest average gap. In fact, our robustified POMO also gains the best solutions for most instances. Detailed results of respective instances are provided in Table IV and Table V.

To summarize, our RET algorithm can favorably robustify POMO, and enhance it comprehensively to attain a good balance between defensive performance and generalizability.

E. Ablation Study

We conduct ablation studies on critical hyperparameters and technical choices in the proposed method. All experiments are done on instances of size 100, with $\omega = 1/6$ in our attack method and $\omega = 1/4$ in RET algorithm if without any specifications. The other configurations are kept the same as presented in subsection V-A.

1) Ablation study on the impact of instance augmentation: We find that instance augmentation plays an important role in resisting the attack from hard instances. We use $\omega = 1/4$ to train ASN models for deep models with and without augmentation, which then generate 1000 hard instances to evaluate optimality gaps, respectively. As illustrated in Figure 5, the decline in deep models' performance is generally alleviated when $\times 8$ instance augmentation is adopted. Our results suggest that the instance augmentation is necessary for POMO-like deep models to keep better performance when faced with hard instances. However, our attack method (especially with $\omega = 1/6$) can still largely degrade deep models even when they are equipped with instance augmentation, which have been verified in subsection V-C.

Instance	Opt.	POMO		POMO_H		POMO_A		POMO_G		POMO_D		POMO_R	
		Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
berlin52	7542	7543	0.01%	7547	0.07%	7543	0.01%	7543	0.01%	7585	0.57%	**7542**	**0.00%**
st70	675	675	0.00%	677	0.30%	675	0.00%	676	0.15%	678	0.44%	**675**	**0.00%**
rat99	1211	1234	1.90%	1259	3.96%	1264	4.38%	1270	4.87%	1220	0.74%	**1214**	**0.25%**
rd100	7910	7910	0.00%	7965	0.70%	8009	1.25%	7924	0.18%	8120	2.65%	**7910**	**0.00%**
kroA100	21282	21367	0.40%	21645	1.71%	21647	1.72%	21458	0.83%	21442	0.75%	**21343**	**0.29%**
kroB100	22141	**22216**	**0.34%**	22488	1.57%	22358	0.98%	22308	0.75%	22418	1.25%	22293	0.69%
kroC100	20749	20785	0.17%	21033	1.37%	21142	1.89%	20876	0.61%	20820	0.34%	**20749**	**0.00%**
kroD100	21294	21472	0.84%	21848	2.60%	21958	3.12%	21548	1.19%	**21389**	**0.45%**	21672	1.78%
kroE100	22068	22165	0.44%	22411	1.55%	22611	2.46%	22256	0.85%	22328	1.18%	**22165**	**0.44%**
pr124	59030	59390	0.61%	59812	1.32%	59602	0.97%	**59030**	**0.00%**	59030	0.00%	59088	0.10%
pr136	96772	97804	1.07%	99901	3.23%	99583	2.91%	**97359**	**0.61%**	97825	1.09%	98127	1.40%
kroA150	26524	26707	0.69%	27511	3.72%	27078	2.09%	27054	2.00%	27095	2.15%	**26657**	0.50%
kroB150	26130	26435	1.17%	26926	3.05%	26972	3.22%	26514	1.47%	26819	2.64%	**26328**	**0.76%**
u159	42080	42512	1.03%	43159	2.56%	43050	2.31%	42652	1.36%	42611	1.26%	**42501**	**1.00%**
rat195	2323	2502	7.71%	2673	15.07%	2599	11.88%	2526	8.74%	2447	5.34%	**2422**	**4.26%**
tsp225	3919	4094	4.47%	4249	8.42%	4214	7.53%	4063	3.67%	4118	5.08%	**4063**	**3.67%**
pr226	80369	83380	3.75%	84804	5.52%	83803	4.27%	**81736**	**1.70%**	83398	3.77%	82528	2.69%
pr264	49135	55352	12.65%	58970	20.02%	56705	15.41%	**50791**	**3.37%**	56114	14.20%	53284	8.44%
a280	2579	2865	11.09%	2950	14.39%	3014	16.87%	2843	10.24%	**2807**	**8.84%**	2808	8.88%
pr299	48191	54529	13.15%	55390	14.94%	54308	12.69%	52567	9.08%	52039	7.98%	**51991**	**7.89%**

TABLE IV: Result on TSPLIB instances. The bold numbers highlight the best objective values and optimality gaps.

Instance	Opt.	POMO		POMO_H		POMO_A		POMO_G		POMO_D		POMO_R	
		Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
X-n101-k25	27591	29282	6.13%	29809	8.04%	29586	7.23%	29725	7.73%	30186	9.41%	**29070**	**5.36%**
X-n115-k10	12747	13877	8.86%	13843	8.60%	13699	7.47%	13718	7.62%	15123	18.64%	**13525**	**6.10%**
X-n125-k30	55539	58412	5.17%	**58142**	**4.69%**	58209	4.81%	58545	5.41%	58353	5.07%	58176	4.75%
X-n143-k7	15700	16382	4.34%	16294	3.78%	16417	4.57%	16700	6.37%	16911	7.71%	**16223**	**3.33%**
X-n148-k46	43448	47613	9.59%	47403	9.10%	47528	8.77%	46773	7.65%	47865	10.17%	**46523**	**7.08%**
X-n172-k51	45607	50351	10.40%	51270	12.42%	51388	12.68%	50869	11.54%	50950	11.72%	**49524**	**8.59%**
X-n176-k26	47812	52889	10.62%	52564	9.94%	**52042**	**8.85%**	52253	9.29%	53446	11.78%	52263	9.31%
X-n181-k23	25569	26969	5.48%	26829	4.93%	26793	4.79%	26566	3.90%	27126	6.09%	**26414**	**3.30%**
X-n195-k51	44225	50296	13.73%	50573	14.35%	50275	13.68%	49445	11.80%	50569	14.34%	**49106**	**11.04%**
X-n200-k36	58578	62094	6.00%	62401	6.53%	61859	5.60%	61916	5.70%	**61591**	**5.14%**	61713	5.35%
X-n204-k19	19565	**20974**	**7.20%**	21417	9.47%	21258	8.65%	21292	8.83%	21563	10.21%	21430	9.53%
X-n219-k73	117595	122204	3.92%	120890	2.80%	**120326**	**2.32%**	121186	3.05%	121542	3.36%	121865	3.63%
X-n223-k34	40437	43794	8.30%	43471	7.50%	43293	7.06%	**43010**	**6.36%**	44273	9.49%	43151	6.71%
X-n242-k48	82751	89184	7.77%	89597	8.27%	88477	6.92%	88303	6.71%	88251	6.65%	**87901**	**6.22%**
X-n251-k28	38684	41355	6.91%	41470	7.20%	**40982**	**5.94%**	41410	7.05%	41658	7.69%	41260	6.66%
X-n261-k13	26558	**28776**	**8.35%**	28849	8.63%	28955	9.03%	29118	9.64%	29847	12.38%	29739	11.98%
X-n266-k58	75478	83616	10.78%	84612	12.10%	82826	9.74%	**82185**	**8.89%**	82706	9.58%	82657	9.51%
X-n270-k35	35291	39952	13.21%	40171	13.83%	38695	9.65%	39034	10.61%	39820	12.83%	**38604**	**9.39%**
X-n289-k60	95151	105343	9.79%	105165	9.60%	104741	10.08%	105238	10.60%	104618	9.95%	**104149**	**8.54%**
X-n294-k50	47161	53937	14.37%	54163	14.85%	**53011**	**12.40%**	54074	14.66%	54985	16.59%	53530	13.51%

TABLE V: Result on CVRPLIB (Set-X) instances. The bold numbers highlight the best objective values and optimality gaps.

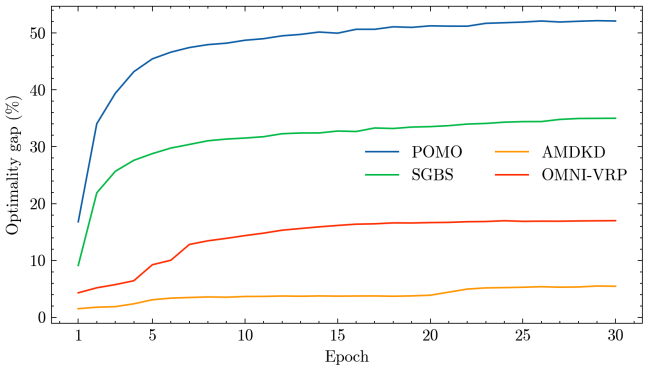

Fig. 6: Training curves for different environment models.

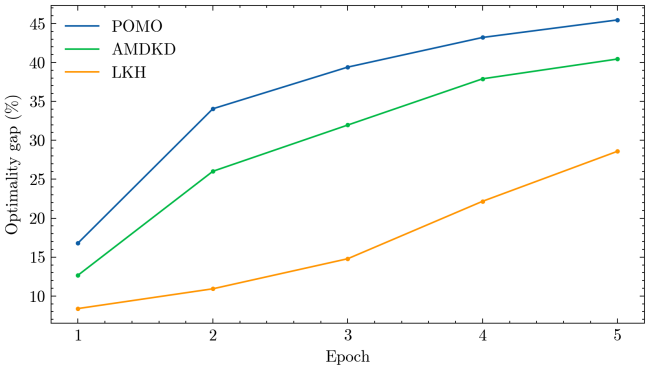

Fig. 7: Training curves with different HMN models.

Additionally, Figure 2 also reveals that distributions of hard instances for different deep models (with instance augmentation) are symmetrical, implying that the instance augmentation transforms instances through symmetry. Accordingly, the proposed ASN is capable of producing model-specific (symmetrical) hard instances in an adaptive manner, which degrade deep models as much as possible.

2) Ablation study on training epochs for attack: In order to strike a balance between efficiency and performance, we only use 5 epochs to train the ASN in all of our main experiments.

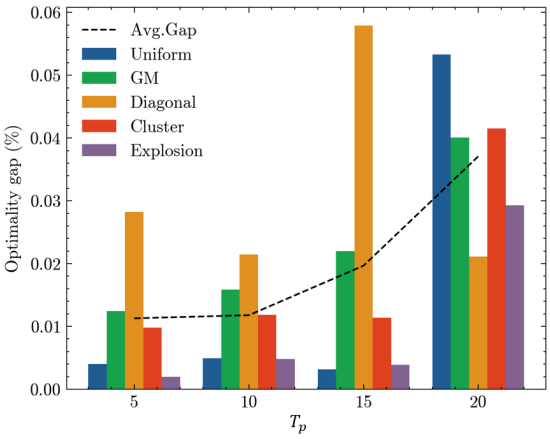

Fig. 8: The cross-distribution performance of RET-trained models with different value of T_p.

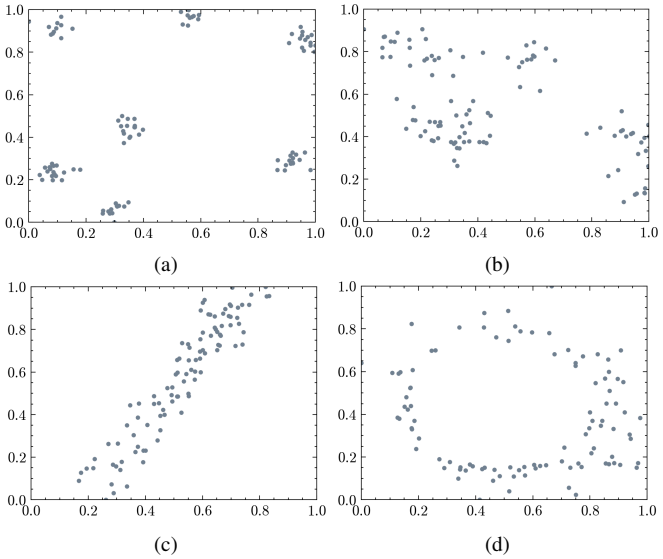

Fig. 9: TSP instances of size 100 sampled from (a) Gaussian mixture distribution; (b) Cluster distribution; (c) Diagonal distribution; (d) Explosion distribution.

Here, we extend the training duration to 30 epochs and show how the performance of the environment model varies when the ASN is trained by more data. In Figure 6, we show how the optimality gaps of different deep models change with the number of training epochs. It is clear that the convergence slows down after the fifth epoch for most deep models. Meanwhile, our attack method is consistently able to obtain more hard instances if we continue to train the ASN.

3) Ablation study on the structure of HMN: Figure 7 depicts the variation of optimality gaps when different solvers are used as HMN for attack. While LKH and AMDKD are two stronger solvers compared to POMO, the deployment with POMO leads to faster convergence and better performance. This empirically verifies that we do not need to obtain the optimal solution for training the ASN. As $M^{'}(\tilde{\mathcal{G}})$ represents the lower bound of the ground-truth optimality gap, it is intuitive that the instances generated by using a weaker HMN model would also be hard, in comparison to a strong solver. In other words, if the lower bound of the optimality gap increases, its ground-truth value also increases, whereas the reverse is not necessarily true.

4) Ablation study on T_p for RET: We increase the parameter T_p from 5 to 20 by a step of 5, and execute RET with each value, respectively. Subsequently, we evaluate the trained models using the identical dataset for evaluating zero-shot generalizability in subsection V-D, with the results presented in Figure 8. As shown, the overall performance of RET diminishes with the increase in training epochs per round. It is primarily attributed to the fact that the evaluation is conducted on manually specified distributions, while the HMN is trained on the generated instances that do not belong to any of these distributions. Thus, training much on the hard instances inevitably deteriorates the generalizability of HMN in solving simpler instances from the specified distributions. On the other hand, the performance of HMN on a specific distribution exhibits a non-linear relationship along the variation of T_p. Such non-linearity stems from the adaptive nature of the ASN, which consistently endeavors to generate hard instances that reflect worst-case scenarios. Since there is no direct correlation between the underlying worst-case scenarios and the manually specified distribution used in evaluation, increasing T_p in an effort to generate (more) hard instances could not yield a significant change of the performance on a specific distribution.

VI. Conclusion

This paper presents a generic framework to generate hard instances and attain robust VRP solutions. Given a pretrained deep model, we alternately train the ASN to sample hard sub-instances from random larger instances, and the HMN to gauge the hardness of the sampled instances. With the objective to maximize the hardness, the trained ASN effectively degrades various deep models for VRPs by generating hostile instances. In this way, we discover model-specific distributions of hard instances, reflecting the worst-case routing scenarios. In addition, we propose the RET algorithm to robustify the deep model and show its effectiveness in enhancing both robustness and zero-shot generalizability. In future, more advanced continual learning techniques can be introduced within RET to further improve its performance, enabling the deep model to learn from hard instances while memorizing the data used for previous training.

Appendix A
Different Distributions and Detailed Generalization Performance

We illustrate the distributions used in subsection V-D in Figure 9. These distributions are defined as follows: (1) **Gaussian mixture distribution**: It is defined by two hyperparameters, i.e., the number of clusters c and the scale l. The coordinates of center nodes for each cluster is firstly generated, and other nodes in each cluster are then sampled following the multivariate Gaussian distribution. In particular, we use $c = 7$ and $l = 50$ to generate corner samples. (2) **Cluster distribution**: The nodes are grouped into N_c clusters, with the quantity of clusters being randomly chosen from

	Method	Uniform			GM			Diagonal			Cluster			Explosion			Avg. Gap
		Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	
TSP	LKH3	7.77	-	8.26m	4.26	-	20.87m	3.24	-	19.65m	5.06	-	16.24m	5.41	-	7.01m	-
	POMO	7.78	0.14%	4.74s	4.37	2.54%	4.74s	3.64	12.33%	4.76s	5.15	1.78%	4.75s	5.42	0.22%	4.75s	3.40%
	POMO_H	7.85	1.04%	4.75s	4.47	4.79%	4.75s	3.68	13.59%	4.75s	5.24	3.58%	4.76s	5.50	1.55%	4.75s	4.91%
	POMO_A	7.84	0.95%	4.75s	4.47	4.84%	4.75s	3.66	13.22%	4.75s	5.24	3.52%	4.76s	5.48	1.36%	4.75s	4.78%
	POMO_G	7.81	0.56%	4.76s	4.27	0.23%	4.75s	3.50	8.14%	4.76s	5.13	1.29%	4.76s	5.43	0.34%	4.76s	2.11%
	POMO_D	7.92	1.89%	4.75s	4.42	3.69%	4.76s	3.24	0.08%	4.76s	5.20	2.77%	4.76s	5.50	1.67%	4.76s	2.02%
	POMO_R	7.80	0.40%	4.76s	4.31	1.24%	4.75s	3.33	2.82%	4.76s	5.11	0.98%	4.75s	5.42	0.20%	4.74s	1.13%
CVRP	LKH3	15.58	-	1.56h	13.88	-	1.52h	11.94	-	2.06h	13.42	-	1.65h	14.37	-	1.51h	-
	POMO	15.76	1.14%	5.87s	14.19	2.23%	5.84s	12.38	3.70%	6.29s	13.71	2.13%	5.91s	14.48	0.75%	5.78s	2.00%
	POMO_H	15.81	1.47%	5.82s	14.25	2.61%	5.80s	12.22	2.38%	5.77s	13.75	2.45%	5.78s	14.56	1.31%	5.77s	2.04%
	POMO_A	15.81	1.45%	5.82s	14.26	2.70%	5.81s	12.23	2.45%	5.80s	13.74	2.40%	5.79s	14.54	1.18%	5.79s	2.04%
	POMO_G	15.83	1.59%	5.80s	14.10	1.60%	5.77s	12.22	2.35%	5.77s	13.70	2.09%	5.78s	14.52	1.07%	5.77s	1.74%
	POMO_D	16.11	3.38%	5.83s	14.29	2.90%	5.82s	11.87	-0.60%	5.78s	13.82	2.94%	5.80s	14.81	3.04%	5.81s	2.33%
	POMO_R	15.82	1.52%	5.80s	14.17	2.05%	5.78s	12.26	2.73%	6.16s	13.71	2.16%	5.78s	14.51	1.00%	5.78s	1.89%

TABLE VI: Evaluation of RET on TSP100 and CVRP100 tasks. POMO_G and POMO_D denote the fine-tuned POMO model on GM and diagonal distribution. POMO_H denotes the model trained with HAM. POMO_R denotes the model trained with RET algorithm.

the set of integers $\{2, 3, ..., 9\}$. The unit square is discretized to a 1000×1000 grid. As center nodes in each cluster are independently generated following Uniform distribution, the probability of a node locating at coordinate p is defined as $\sum_{n=0}^{N_c} \exp(-c_{p,n}/40)$, where n denotes the center node of each cluster and $c_{p,n}$ denotes the distance between p and n. (3) **Diagonal distribution**: One of the two diagonals of a rectangle is uniformly chosen for generation. For each instance, two noises are sampled from the Uniform distribution $U(-t, t)$ with t sampled from $U(0.05, 0.2)$, in order to transform the horizontal and vertical of nodes. (4) **Explosion distribution**: The uniformly sampled nodes are mutated by simulating an explosion. For each instance, the center v_c of the explosion is randomly selected, and all nodes v_i within the explosion radius $R = 0.5$ is moved away from v_c by $v_i = v_c + \frac{v_c - v_i}{\|v_c - v_i\|}(R + e)$, where $e \sim \text{Exp}(\lambda = \frac{1}{10})$ is a random variable.

We report the average solving time and (near-)optimality gaps for each tested model. The performance on different distributions are shown in Table VI. We observe that fine-tuning the model on a distribution can make it perform well on this specific distribution, but could weaken its ability to generalize to other unseen distributions. On the contrary, the RET-trained model is able to achieve balanced performance across distributions, even if we did not explicitly take these distributions into account in the training process.

REFERENCES

[1] J. Duan, Z. He, and G. G. Yen, "Robust multiobjective optimization for vehicle routing problem with time windows," *IEEE Transactions on Cybernetics*, vol. 52, no. 8, pp. 8300–8314, 2022.

[2] P. C. Pop, O. Cosma, C. Sabo, and C. P. Sitar, "A comprehensive survey on the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 314, no. 3, pp. 819–835, 2024.

[3] Y.-H. Jia, Y. Mei, and M. Zhang, "A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 855–10 868, 2022.

[4] H. Lu, Z. Li, R. Wang, Q. Ren, X. Li, M. Yuan, J. Zeng, X. Yang, and J. Yan, "ROCO: A general framework for evaluating robustness of combinatorial optimization solvers on graphs," in *International Conference on Learning Representations*, 2023.

[5] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *ArXiv*, vol. abs/1906.01227, 2019.

[6] Y. Jiang, Y. Wu, Z. Cao, and J. Zhang, "Learning to solve routing problems via distributionally robust optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 9786–9794.

[7] S. Li, Z. Yan, and C. Wu, "Learning to delegate for large-scale vehicle routing," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 198–26 211, 2021.

[8] J. Li, Y. Ma, R. Gao, Z. Cao, A. Lim, W. Song, and J. Zhang, "Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem," *IEEE Transactions on Cybernetics*, vol. 52, no. 12, pp. 13 572–13 585, 2022.

[9] Q. Hou, J. Yang, Y. Su, X. Wang, and Y. Deng, "Generalize learned heuristics to solve large-scale vehicle routing problems in real-time," in *International Conference on Learning Representations*, 2023.

[10] J. Bi, Y. Ma, J. Wang, Z. Cao, J. Chen, Y. Sun, and Y. M. Chee, "Learning generalizable models for vehicle routing problems via knowledge distillation," in *Advances in Neural Information Processing Systems*, 2022.

[11] M. Kim, J. SON, H. Kim, and J. Park, "Scale-conditioned adaptation for large scale combinatorial optimization," in *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2022.

[12] A. Bdeir, J. K. Falkner, and L. Schmidt-Thieme, "Attention, filling in the gaps for generalization in routing problems," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2022.

[13] J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang, "Towards omni-generalizable neural methods for vehicle routing problems," in *International Conference on Machine Learning*, 2023.

[14] S. Manchanda, S. Michel, D. Drakulic, and J.-M. Andreoli, "On the generalization of neural combinatorial optimization heuristics," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2022.

[15] S. Geisler, J. Sommer, J. Schuchardt, A. Bojchevski, and S. Günnemann, "Generalization of neural combinatorial solvers through the lens of adversarial robustness," in *International Conference on Learning Representations*, 2022.

[16] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-end constrained optimization learning: A survey," in *International Joint Conference on Artificial Intelligence*, 2021, pp. 4475–4482.

[17] D. Ireland and G. Montana, "LeNSE: Learning to navigate subgraph embeddings for large-scale combinatorial optimisation," in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, 2022, pp. 9622–9638.

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

[19] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *International conference on learning representations*, 2019.

[20] P. R. d O Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Asian Conference on Machine Learning*, 2020, pp. 465–480.

[21] 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*, vol. 33, no. 9, pp. 5057–5069, 2022.

[22] B. Hudson, Q. Li, M. Malencia, and A. Prorok, "Graph neural network guided local search for the traveling salesperson problem," in *International Conference on Learning Representations*, 2021.

[23] 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 trans-

former," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 096–11 107, 2021.
[24] L. Xin, W. Song, Z. Cao, and J. Zhang, "Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 7472–7483.
[25] R. Wang, Z. Hua, G. Liu, J. Zhang, J. Yan, F. Qi, S. Yang, J. Zhou, and X. Yang, "A bi-level framework for learning to solve combinatorial optimization on graphs," vol. 34, 2021, pp. 21 453–21 466.
[26] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li, "Deepaco: Neural-enhanced ant systems for combinatorial optimization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
[27] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByxBFsRqYm
[28] 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, 2020, pp. 21 188–21 198.
[29] F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang, "Neural combinatorial optimization with heavy decoder: Toward large scale generalization," in *Advances in Neural Information Processing Systems*, 2023.
[30] Z. Sun and Y. Yang, "Difusco: Graph-based diffusion solvers for combinatorial optimization," *arXiv preprint arXiv:2302.08224*, 2023.
[31] C. Wang, Y. Yang, O. Slumbers, C. Han, T. Guo, H. Zhang, and J. Wang, "A game-theoretic approach for improving generalization ability of tsp solvers," *arXiv preprint arXiv:2110.15105*, 2021.
[32] M. Lisicki, A. Afkanpour, and G. W. Taylor, "Evaluating curriculum learning strategies in neural combinatorial optimization," in *NeurIPS 2020 Workshop on Learning Meets Combinatorial Algorithms*, 2020.
[33] L. Xin, W. Song, Z. Cao, and J. Zhang, "Generative adversarial training for neural combinatorial optimization models," 2022. [Online]. Available: https://openreview.net/forum?id=9vsRT9mc7U
[34] Z. Zhang, Z. Zhang, X. Wang, and W. Zhu, "Learning to solve travelling salesman problem with hardness-adaptive curriculum," in *AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 9136–9144.
[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[36] 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*, 2016.
[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37. PMLR, 2015, pp. 448–456.
[38] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.
[39] X. Lin, Z. Yang, and Q. Zhang, "Pareto set learning for neural multi-objective combinatorial optimization," in *International Conference on Learning Representations*, 2022.
[40] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.
[41] J. Choo, Y.-D. Kwon, J. Kim, J. Jae, A. Hottung, K. Tierney, and Y. Gwon, "Simulation-guided beam search for neural combinatorial optimization," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 8760–8772.
[42] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[43] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," in *Advances in Neural Information Processing Systems*, 2021.
[44] G. Gutin, A. Yeo, and A. Zverovich, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp," *Discrete Applied Mathematics*, vol. 117, no. 1, pp. 81–86, 2002.
[45] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II, "An analysis of several heuristics for the traveling salesman problem," *SIAM journal on computing*, vol. 6, no. 3, pp. 563–581, 1977.
[46] J. Bossek, P. Kerschke, A. Neumann, M. Wagner, F. Neumann, and H. Trautmann, "Evolving diverse tsp instances by means of novel and creative mutation operators," in *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, ser. FOGA '19, 2019, p. 58–71.
[47] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.
[48] 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.