7-2023

Towards omni-generalizable neural methods for vehicle routing problems

Jianan ZHOU

Yaoxin WU

Wen SONG

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

Jie ZHANG

Towards Omni-generalizable Neural Methods for Vehicle Routing Problems

Jianan Zhou [1] **Yaoxin Wu** [2] **Wen Song** [3] **Zhiguang Cao** [4] **Jie Zhang** [1]

Abstract

Learning heuristics for vehicle routing problems (VRPs) has gained much attention due to the less reliance on hand-crafted rules. However, existing methods are typically trained and tested on the same task with a fixed size and distribution (of nodes), and hence suffer from limited generalization performance. This paper studies a challenging yet realistic setting, which considers generalization across both size and distribution in VRPs. We propose a generic meta-learning framework, which enables effective training of an initialized model with the capability of fast adaptation to new tasks during inference. We further develop a simple yet efficient approximation method to reduce the training overhead. Extensive experiments on both synthetic and benchmark instances of the traveling salesman problem (TSP) and capacitated vehicle routing problem (CVRP) demonstrate the effectiveness of our method. The code is available at: https://github.com/RoyalSkye/Omni-VRP.

1. Introduction

Combinatorial optimization problems (COPs) are of great importance in computer science and operation research. The exact methods suffer from the scalability issue due to the NP-hardness, while the heuristic ones need substantial hand-crafted rules and domain expertise for each specific problem. Recently, the *neural method*, which leverages machine learning (ML) to automatically learn or discover heuristics for a wide range of COPs, has gained much attention (Bengio et al., 2021). By exploiting the underlying pattern among a group of COP instances, the neural method has the potential to reduce computational efforts while achieving desirable solution quality. In this paper, we focus on *vehicle routing problems* (VRPs), which is a class of canonical NP-hard COPs with wide applications in transportation (Garaix et al., 2010; Zhou et al., 2023; Wu et al., 2023) and logistics (Cattaruzza et al., 2017; Konstantakopoulos et al., 2022). The neural methods for VRPs usually employ advanced deep models (e.g., pointer network (Vinyals et al., 2015), attention mechanism (Vaswani et al., 2017) and graph neural network (Bresson & Laurent, 2017)) to learn heuristics for route construction or improvement with supervised learning or reinforcement learning. They have achieved competitive or even superior performance to the conventional heuristics.

However, most of the neural methods are trained and tested on the same task with a fixed size and distribution, and thus suffer from poor generalization. For example, the popular attention-based models (Kool et al., 2018; Kwon et al., 2020) are only trained and tested on instances of fixed size (e.g., 100), with node coordinates sampled from the uniform distribution. The performance of the learned heuristic drastically decreases when it is applied to an unseen task during training (see POMO* in Table 1). This generalization issue severely hinders the application of these neural methods in practice. A simple measure for improvement is to train them on diverse data. However, covering the whole problem space is intractable (e.g., the "Catch-22" for NP-hard problems (Yehuda et al., 2020)). Thus how to effectively learn from diverse data for VRPs is still a challenging problem.

While some attempts have been made to tackle the generalization issue of neural methods for VRPs, most of them solely focus on either size (Lisicki et al., 2020; Bdeir et al., 2022; Kim et al., 2022b) or distribution (Zhang et al., 2022; Jiang et al., 2022; Wang et al., 2022; Geisler et al., 2022; Bi et al., 2022). We argue that it is more realistic to simultaneously consider the generalization of size and distribution, since the real-world VRP instances (e.g., TSPLIB (Reinelt, 1991) and CVRPLIB (Uchoa et al., 2017)) may vary in both. As a promising work, Manchanda et al. (2022) handles this challenging setting by exploiting a meta-learning technique, i.e., Reptile (Nichol et al., 2018), which only relies on the first-order derivatives for training. However, it is still far from satisfaction in that, 1) Reptile is less sample efficient as it typically needs multiple inner-loop updates to incorporate

[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore [2]Department of Information Systems, Eindhoven University of Technology, The Netherlands [3]Institute of Marine Science and Technology, Shandong University, China [4]School of Computing and Information Systems, Singapore Management University, Singapore. Correspondence to: Yaoxin Wu <wyxacc@hotmail.com>, Wen Song <wensong@email.sdu.edu.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

information from higher-order derivatives (see Appendix A) to achieve desirable performance (see Appendix D.2). Moreover, theoretical and empirical evidences of its effectiveness are only demonstrated on the few-shot supervised learning setting (Nichol et al., 2018) rather than the reinforcement learning setting, the latter of which is more favored by the neural methods for VRPs; 2) it simply selects the training task by randomly sampling from the task set in each iteration, which overlooks the training dynamics and fails to fully make use of the diverse data information.

In this paper, we tackle the omni-generalization issue of neural VRP methods by training on diverse tasks, each of which relates to a unique size and distribution. According to the *No Free Lunch Theorems of Machine Learning* (Wolpert & Macready, 1997), it is unrealistic to train a one-size-fits-all model that could perform well on any task. Therefore, we also resort to meta-learning (Vilalta & Drissi, 2002; Hospedales et al., 2021) to learn a good initialized model for fine-tuning afterwards, while bypassing the limitations in Manchanda et al. (2022). Specifically, we propose a generic meta-learning framework, which is model-agnostic and compatible with any model trained with gradient updates. It learns a good initialization of model parameters by performing meta-training on tasks, which are adaptively selected from the training task set via a hierarchical scheduler. The trained model is able to efficiently adapt to new tasks only using limited data during inference. Despite being effective, it needs the second-order derivatives to perform meta-model updates, making it computationally expensive when training on instances of large sizes. Therefore, we further develop a simple yet efficient approximation method by early stopping the usage of second-order derivatives, and only leveraging the first-order ones afterwards.

Our contributions are summarized as follows. 1) We study a challenging yet realistic setting for neural VRP methods by considering the omni-generalization across both size and distribution. 2) We propose a generic meta-learning framework, where we leverage a second-order technique to enable effective learning of an initialized model with the capability of efficient adaptation to new tasks only using limited data during inference. 3) To reduce the meta-training cost, we develop a simple yet efficient approximation method, which performs comparable to the one with full second-order derivatives. 4) We evaluate the effectiveness of our method on the traveling salesman problem (TSP) and capacitated vehicle routing problem (CVRP) by meta-training POMO (Kwon et al., 2020) and L2D (Li et al., 2021). The experimental results demonstrate that our method could improve the omni-generalization of the base models even on the zero-shot setting. We also observe consistent superiority on the few-shot settings, and on the classical benchmark datasets such as TSPLIB and CVRPLIB.

2. Related Work

Neural Methods for VRPs. Most of recent neural methods for VRPs could be divided into two categories: 1) *Learning Construction Heuristics:* the solution is constructed sequentially or in a one-shot manner by the learned heuristic without iterative modifications. Vinyals et al. (2015) proposes the Pointer Network (Ptr-Net) to solve TSP with supervised learning. Subsequent works train Ptr-Net using reinforcement learning to solve TSP (Bello et al., 2017) and CVRP (Nazari et al., 2018). Kool et al. (2018) introduces the attention model (AM) based on the Transformer architecture (Vaswani et al., 2017) to solve a wide range of COPs including TSP and CVRP. Kwon et al. (2020) further proposes the policy optimization with multiple optima (POMO), which improves upon AM by exploiting solution symmetries. Besides Ptr-Net and attention-based models, graph neural networks are also exploited to solve VRPs (Dai et al., 2017; Joshi et al., 2019). Other works have also been proposed to improve upon them (Ma et al., 2019; Kwon et al., 2021; Xin et al., 2021a; Kim et al., 2022a). 2) *Learning Improvement Heuristics:* an initial complete solution is iteratively refined by the learned heuristic until a termination condition is satisfied. In this line of research, the classical local search methods (e.g., 2-opt (Croes, 1958) or large neighborhood search (LNS) (Shaw, 1998)) or (part of) specialized heuristic solvers for VRPs (e.g., Lin-Kernighan-Helsgaun (LKH) (Helsgaun, 2000; 2017)) are usually exploited (Chen & Tian, 2019; Lu et al., 2020; Hottung & Tierney, 2020; d O Costa et al., 2020; Wu et al., 2021; Wang et al., 2021; Ma et al., 2021; Xin et al., 2021b; Kim et al., 2021; Hudson et al., 2022). In general, the improvement heuristics could achieve better performance than the construction ones, but at the expense of much longer inference time. Besides VRPs, neural methods have also been applied to solve other COPs, such as the job shop scheduling problem (JSSP) (Zhang et al., 2020), maximal independent set (MIS) (Dai et al., 2017; Ahn et al., 2020) and boolean satisfiability (SAT) (Selsam et al., 2019). In this paper, we mainly focus on the neural methods for VRPs.

Generalization of Neural Methods. Previous works mainly train and test on instances with the same size and distribution, which results in poor generalization performance (Joshi et al., 2021; Liu et al., 2022). Recently, some works attempt to improve the generalization capability of neural methods for VRPs, which could be further divided into two classes: 1) *Size Generalization:* the aim is to generalize the learned heuristics to instances smaller or larger[1] than the training ones. Lisicki et al. (2020) trains a model on multiple sizes via the curriculum learning strategy (Bengio et al., 2009). Kim et al. (2022b) and Bdeir et al. (2022)

[1]The community cares more about the larger instances since they are usually much harder than the smaller ones.

study the size generalization upon the attention-based models by improving their structures, such as incorporating a scale conditioned network and a sparse dynamic attention, respectively. We also would like to mention another line of research that deals with the *scalability*, which is less related to the generalization issue studied in this paper. Specifically, they mainly rely on the heat-map (Joshi et al., 2019; Fu et al., 2021; Qiu et al., 2022; Sun & Yang, 2023) or decomposition (Li et al., 2021; Hou et al., 2023) so as to scale up to larger instances. 2) *Distribution Generalization:* the aim is to generalize the learned heuristics to instances sampled from various (unseen) distributions. Some works augment the training instances with diverse distributions either by jointly training an instance generator (Wang et al., 2022) or adversarial training (Zhang et al., 2022; Geisler et al., 2022). Other than the data perspective, Jiang et al. (2022) and Bi et al. (2022) exploit the group distributionally robust optimization (Sagawa et al., 2020) and knowledge distillation (Hinton et al., 2015) to improve the distribution generalization, respectively.

Although some efforts have been made either for the size or distribution generalization, rare works consider both of them. Manchanda et al. (2022) first studies this setting using Reptile (Nichol et al., 2018). It gradually updates the meta-model towards the task-specific model on each task such that the learned meta-model could serve as a good initialization for fine-tuning during inference. In this paper, we also focus on this challenging yet realistic problem setting.

3. Preliminaries

In this section, we introduce the problem statement for VRPs and the Markov Decision Process (MDP) formulation for constructing solutions to VRP instances. Without loss of generality, we define a VRP instance of size n over a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $v_i \in \mathcal{V}$ ($\mathcal{V} = \{v_i\}_{i=1}^n$) represents the node (e.g., customer in TSP) and $e(v_i, v_j) \in \mathcal{E}$ represents the edge between node v_i and v_j. The solution (i.e., tour) τ is represented as a sequence of nodes in $\mathcal{V}$. In this paper, we consider the Euclidean VRPs with the cost function $c(\cdot)$ defined as the total length of the tour. The objective of VRPs is to find the optimal tour τ^* with the *minimal* cost:

$$\tau^* = \arg\min_{\tau \in S} c(\tau|\mathcal{G}), \tag{1}$$

where $\mathcal{S}$ is the discrete search space that contains all the feasible tours, subject to the problem-specific constraints. Specifically, a feasible tour in TSP should visit each (customer) node exactly once and return to the starting node in the end. On top of TSP, a depot node v_0 is introduced in CVRP, where each customer node is featured by a demand δ_i, and a capacity limit Q is set for each vehicle. A tour in CVRP consists of multiple sub-tours, each of which represents a vehicle starting from the depot, visiting a subset of nodes in $\mathcal{V}$ and returning to the depot. It is feasible if each customer node is visited exactly once and the total demand in each sub-tour does not exceed the capacity limit Q.

Neural construction methods formulate the solving process of a VRP instance $\mathcal{G}$ as a MDP, where they typically parameterize the policy by an encoder-decoder structured neural network to learn node selection for constructing a solution. The encoder in the policy network outputs a global representation of the instance, which, with the representation of the context (e.g., the partial tour in construction), captures the current state. The decoder takes the global and context representations as inputs to compute the probabilities of nodes (i.e., actions) to be visited. The node is selected sequentially until a complete tour τ is constructed. Hence, the probability of the tour is factorized via the chain rule as:

$$p_\theta(\tau|\mathcal{G}) = \prod_{t=1}^{T} p_\theta(\pi_\theta(t)|\pi_\theta(<t), \mathcal{G}), \tag{2}$$

where $\pi_\theta(t)$ and $\pi_\theta(<t)$ are the selected node and the current partial solution at time step t, respectively; T denotes the number of total steps. The reward is defined as the negative cost of a tour, i.e., $\mathcal{R} = -c(\tau|\mathcal{G})$. To train the policy network, the REINFORCE algorithm (Williams, 1992) is commonly used to estimate the gradient of the expected reward $\mathcal{L}(\theta|\mathcal{G}) = \mathbb{E}_{p_\theta(\tau|\mathcal{G})} c(\tau)$ such that:

$$\nabla_\theta \mathcal{L}(\theta|\mathcal{G}) = \mathbb{E}_{p_\theta(\tau|\mathcal{G})}[(c(\tau) - b(\mathcal{G}))\nabla_\theta \log p_\theta(\tau|\mathcal{G})], \tag{3}$$

where $b(\cdot)$ is a baseline function to reduce gradient variance.

4. Methodology

In this section, we introduce a generic meta-learning framework to tackle the omni-generalizable issue of neural methods across both size and distribution in VRPs. We further develop a simple yet efficient approximation method to promote the training efficiency. Without loss of generality, we present our method by taking the meta-training of POMO (Kwon et al., 2020) on TSP as an example.

4.1. Meta-Learning Framework

We define a task $\mathcal{T}(n, d) \sim p(\mathcal{T})$ as a class of instances with the same size $n \in \mathcal{N}$ and distribution $d \in \mathcal{D}$, where $\mathcal{N} = [n_{\min}, n_{\max}]$ is the range of problem sizes; $\mathcal{D} = \{\mathcal{D}_j\}_{j=1}^{|\mathcal{D}|}$ is a set of distributions; $p(\mathcal{T})$ is the underlying task distribution. For notation simplicity, we use $\mathcal{T}_i$ to represent a task. Inspired by MAML (Finn et al., 2017), we propose a generic meta-learning framework to improve the generalization capability of neural methods for VRPs, which is model-agnostic and compatible with any model trained with gradient updates. The framework is illustrated in Figure 1, and we elaborate its key components below.

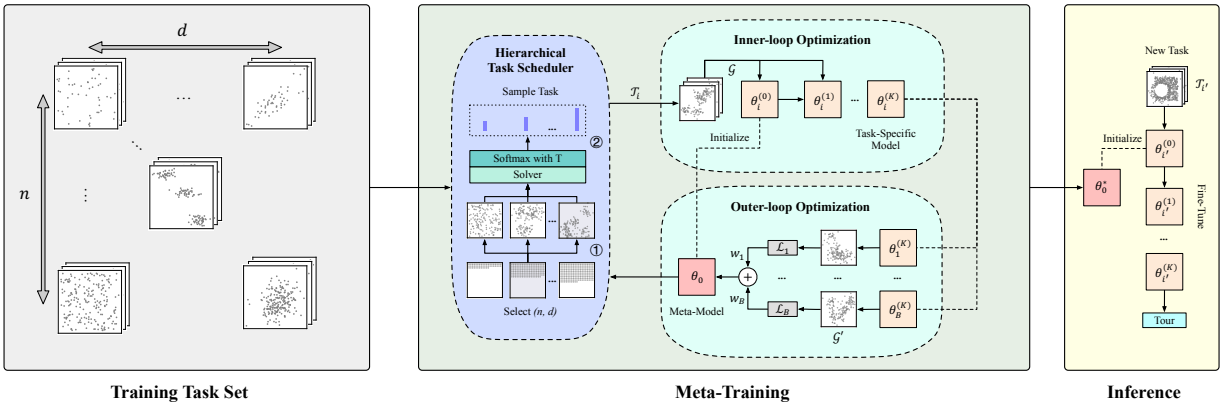


Figure 1. The illustration of the proposed framework. The training task set consists of tasks with diverse sizes $n \sim \mathcal{N}$ and distributions $d \sim \mathcal{D}$. In each iteration, the hierarchical task scheduler adaptively selects a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$ for meta-training, which consists of a pair of inner-loop and outer-loop optimization. During inference, given a new task $\mathcal{T}_{i'}$, the trained meta-model θ_0^* is used to initialize the task-specific model $\theta_{i'}^{(0)}$, which is then adapted to $\mathcal{T}_{i'}$ by further taking K gradient steps with a limited number of instances.

Meta-Training. In the meta-learning framework, we aim to train a meta-model θ_0, which as a good initialized model can be efficiently adapted to new tasks during inference. Formally, we define the meta-objective as follows:

$$\theta_0^* = \arg\min_{\theta_0} \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})} \mathbb{E}_{\mathcal{G} \sim \mathcal{T}_i} \mathcal{L}_i(\theta_i^{(K)} | \mathcal{G}), \tag{4}$$

where $\theta_i^{(K)}$ is the fine-tuned model after K gradient updates of θ_0 on the task $\mathcal{T}_i$; $\mathcal{L}_i$ is the loss function on the task $\mathcal{T}_i$. In this paper, we use the same loss function (e.g., reinforcement loss) for different tasks. To *directly* optimize this objective, the meta-training procedure comprises the inner-loop and outer-loop optimization at each iteration. The meta-training pseudocode is presented in Algorithm 1.

Inner-loop optimization: It optimizes a task-specific model iteratively, which is similar to the fine-tuning stage during inference. Specifically, given a task $\mathcal{T}_i \sim p(\mathcal{T})$, we initialize the task-specific model by the meta-model, i.e., $\theta_i^{(0)} \leftarrow \theta_0$ (line 5), and adapt it to $\mathcal{T}_i$ by performing K gradient update steps on the training instances (line 6-11). The gradient of the loss function $\mathcal{L}_i$ at the k_{th} step is computed as follows:

$$\nabla_{\theta_i^{(k-1)}} \mathcal{L}_i(\theta_i^{(k-1)}) \leftarrow \frac{1}{M} \sum_{m=1}^{M} \nabla_{\theta_i^{(k-1)}} \mathcal{L}_i(\theta_i^{(k-1)} | \mathcal{G}_m), \tag{5}$$

where $\nabla_{\theta_i^{(k-1)}} \mathcal{L}_i(\theta_i^{(k-1)} | \mathcal{G}_m)$ can be estimated by the REINFORCE algorithm as in Eq. (3).

Outer-loop optimization: It optimizes the meta-model with the objective in Eq. (4). Concretely, for each task $\mathcal{T}_i$, the few-shot generalization performance of the task-specific model $\theta_i^{(K)}$ is evaluated on the validation instances. The meta-gradient (line 13) is obtained as follows:

$$\nabla_{\theta_0} \mathcal{L}_i(\theta_i^{(K)}) \leftarrow \frac{1}{M} \sum_{m=1}^{M} \nabla_{\theta_i^{(K)}} \mathcal{L}_i(\theta_i^{(K)} | \mathcal{G}'_m) \cdot \frac{\partial \theta_i^{(K)}}{\partial \theta_0}. \tag{6}$$

After conducting the inner-loop optimization on a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$, the meta-model θ_0 is then updated once (as in line 16). Intuitively, the inner-loop optimization serves as the task adaption stage, imitating the fine-tuning process during inference, while the outer-loop optimization updates the meta-model with the objective of maximizing the few-shot generalization performance of the task-specific model $\theta_i^{(K)}$. Therefore, after meta-training, we can get a good initialized model θ_0^* with the capability of efficient adaptation to new tasks only using limited data. Note that Eq. (6) needs the second-order derivatives since we expect to get the gradient direction with respect to the meta-model θ_0. We will discuss and analyze its first-order approximation methods later.

Hierarchical Task Scheduler. The aim of a task scheduler is to guide the task selection for the meta-training process, so as to improve the optimization performance. Most existing works (Finn et al., 2017; Raghu et al., 2020; Flennerhag et al., 2022; Manchanda et al., 2022) randomly sample training tasks from the task set with a uniform probability, which assumes all tasks are equally important. It may overlook the training dynamics and cannot make full use of diverse data information. In this paper, we propose a simple yet effective hierarchical task scheduler for VRPs.

During training, we first gradually increase the size of training task following a linear scheduler. At the e_{th} iteration of meta-training, we select tasks of the size $n_e = \lfloor n_{\min} + \min(\frac{e}{E_s}, 1) \cdot (n_{\max} - n_{\min}) \rfloor$, where E_s is the working duration of the scheduler. Then, a probability distribution over the tasks in $\{\mathcal{T}(n_e, d_j)\}_{j=1}^{|\mathcal{D}|}$ is generated for

Algorithm 1 Meta-Training for VRPs

Input: distribution over tasks $p(\mathcal{T})$, number of tasks in a mini-batch B, number of inner-loop updates K, batch size M, step sizes of inner-loop and outer-loop optimization α, β;
Output: meta-model θ_0^*;

1: Initialize meta-model θ_0
2: **while** not done **do**
3: $\{\mathcal{T}_i, w_i\}_{i=1}^B, \leftarrow$ Hierarchical task scheduler
4: **for** $i = 1, \ldots, B$ **do**
5: Initialize task-specific model $\theta_i^{(0)} \leftarrow \theta_0$
6: **for** $k = 1, \ldots, K$ **do**
7: { *// Inner-loop optimization* }
8: Sample training instances $\{\mathcal{G}_m\}_{m=1}^M$ from task $\mathcal{T}_i$
9: Obtain $\nabla_{\theta_i^{(k-1)}} \mathcal{L}_i(\theta_i^{(k-1)})$ using Eq. (5)
10: $\theta_i^{(k)} \leftarrow \theta_i^{(k-1)} - \alpha \nabla_{\theta_i^{(k-1)}} \mathcal{L}_i(\theta_i^{(k-1)})$
11: **end for**
12: Sample validation instances $\{\mathcal{G}'_m\}_{m=1}^M$ from task $\mathcal{T}_i$
13: Obtain $\nabla_{\theta_0} \mathcal{L}_i(\theta_i^{(K)})$ using Eq. (6)
14: **end for**
15: { *// Outer-loop optimization* }
16: $\theta_0 \leftarrow \theta_0 - \beta \sum_{i=1}^B w_i \nabla_{\theta_0} \mathcal{L}_i(\theta_i^{(K)})$
17: **end while**

training task selection, based on their hardness. The optimality gap is an appropriate metric to measure the hardness of each task. However, it is intractable to obtain optimal solutions due to the NP-hardness. Here we use a general VRP solver LKH3 (Helsgaun, 2017) to efficiently[2] obtain near-optimal solutions for a validation set of instances in a one-shot manner. The validation instances are fixed throughout the meta-training process, and hence we only need to run LKH3 once. To avoid overfitting, we only sample M instances from the validation set to calculate the relative gap $g_i = \frac{1}{M} \sum_{m=1}^M \frac{c(\tau_m) - c(\bar{\tau}_m)}{c(\bar{\tau}_m)}$ for each task $\mathcal{T}_i$, where τ_m and $\bar{\tau}_m$ are solutions to the m_{th} instance constructed by the current meta-model and LKH3, respectively. Accordingly, the probability of selecting each task $\mathcal{T}_i$ is defined as:

$$w_i = \frac{\exp(g_i/\eta)}{\sum_{j=1}^{|\mathcal{D}|} \exp(g_j/\eta)}, \tag{7}$$

where η is the temperature to control the entropy of probability distribution, from which the task is sampled. The initial probability distribution is uniform, and is updated by Eq. (7) periodically (e.g., every 100 iterations). After a batch of tasks $\{\mathcal{T}_i\}_{i=1}^B$ is sampled, we normalize their weights such that $\sum_{i=1}^B w_i = 1$. Then, the meta-model is optimized with the weighted sum of their losses $\{w_i \mathcal{L}_i(\theta_i^{(K)})\}_{i=1}^B$ (line 16).

Inference. During inference, given instances sampled from a new task $\mathcal{T}_{i'}$, the trained meta-model θ_0^* could be used to approximate solutions in several ways: 1) *zero-shot:* the solution is directly constructed using the learned policy $\pi_{\theta_0^*}$ with efficient search strategies (e.g., greedy rollout); 2) *few-shot:* similar to the inner-loop optimization as shown in line 6-11 of Algorithm 1, the task-specific model $\theta_{i'}$ is initialized by θ_0^*, and is adapted to the new task $\mathcal{T}_{i'}$ by taking K gradient steps on a small set of instances (different from test instances) sampled from $\mathcal{T}_{i'}$. Then the solution is constructed using the adapted policy; 3) *active search:* the model is adapted to each test instance by learning instance-dependent parameters. Bello et al. (2017) first proposes the general active search that iteratively adjusts the model parameters with the objective of increasing the likelihood of constructing high-quality solutions for each instance. However, it is extremely computationally expensive. Hottung et al. (2022) proposes the efficient active search (EAS) by introducing extra instance-specific parameters (e.g., a MLP layer) for each test instance while fixing the original model parameters. We mainly consider the zero-shot and few-shot settings in our experiments (in Section 5), and only use EAS when evaluating on benchmark instances (in Appendix D.3).

[2]We set the maximum trials of LKH3 to 100 for efficiency.

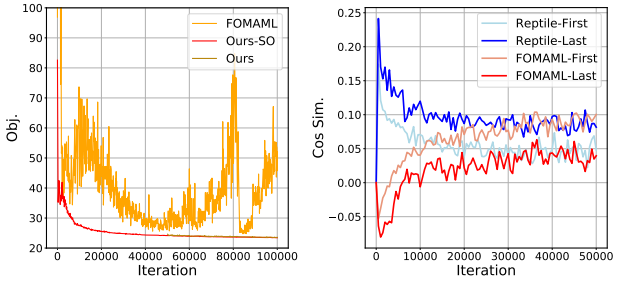


Figure 2. Left panel: the meta-training curves; *Right panel:* the cosine similarity between the second-order derivative and its first-order approximation at the first and last layer of the model.

4.2. First-Order Approximation

The meta-gradient in Eq. (6) involves a gradient through a gradient (i.e., second-order derivative), and therefore it is computational expensive to obtain due to the calculation of Hessian-vector products. To tackle this issue, Finn et al. (2017) proposes a first-order approximation method (i.e., FOMAML) which simply drops the second-order term. The empirical evidence of its effectiveness has been verified in the few-shot supervised learning, while lacking in the more complex reinforcement learning. Specifically, the first-order approximation of meta-model update can be expressed as:

$$\theta_0 \leftarrow \theta_0 - \beta \sum_{i=1}^B w_i \nabla_{\theta_i} \mathcal{L}_i(\theta_i^{(K)}). \tag{8}$$

However, as shown in the left panel of Figure 2, we empirically observe that meta-training POMO from scratch with Eq. (8) may induce fluctuating validation performance. The unstable meta-training may be attributed to the deviation of the first-order approximation (Eq. (8)) from the (ground-truth) gradient direction of the second-order term (line 16 in

Algorithm 1). Intuitively, $\text{sign}(\nabla_{\theta_i}\mathcal{L}_i(\theta_i^{(K)}))$ is the (steepest) descent direction for the task-specific model $\theta_i^{(K)}$, but not necessarily the descent direction for the meta-model θ_0, especially at the *early stage* of meta-training when the optimization tends to be unstable. To justify our hypothesis, we show the cosine similarities of the gradient directions for meta-model updates between our method and others in the right panel of Figure 2. The detailed experimental setups are presented in Appendix D.1. We observe that $\text{sign}(\nabla_{\theta_i}\mathcal{L}_i(\theta_i^{(K)}))$ cannot approximate $\text{sign}(\nabla_{\theta_0}\mathcal{L}_i(\theta_i^{(K)}))$ well (i.e., with a negative cosine similarity) at the early stage of meta-training, while gradually having the positive correlation as the training progresses. Therefore, in order to reduce the computational cost and stabilize the meta-training, we develop a simple yet efficient method by early stopping the usage of second-order derivatives. Specifically, we start the meta-model updates with second-order derivatives, and switch to the first-order ones (i.e., by replacing line 16 with Eq. (8)) when the optimization tends to be stable. Recently, Manchanda et al. (2022) leverages another first-order method called Reptile (Nichol et al., 2018), with the form of the meta-model update as follows:

$$\theta_0 \leftarrow \theta_0 + \beta \sum_{i=1}^{B} w_i(\theta_i^{(K)} - \theta_0). \tag{9}$$

However, it needs multiple inner-loop updates to effectively incorporate information from higher-order derivatives of the loss function so as to achieve satisfactory performance (as shown in Appendix A and D.2), making it less sample efficient. Moreover, similar to Nichol et al. (2018) which observes negative results after applying Reptile to the reinforcement learning setting, we also empirically find its weak performance (see Meta-POMO in Section 5, Appendix D.2 and D.3). In contrast, our method could achieve decent performance only running a single inner-loop update.

5. Experiments

To demonstrate the effectiveness of the proposed framework, we apply it to POMO[3] (Kwon et al., 2020), which is a strong construction-based neural method. We consider two representative VRP problems (i.e., TSP and CVRP). The details of POMO are introduced in Appendix C. Moreover, we also evaluate the generalizability of our method on L2D (Li et al., 2021) as shown in Section 5.3 and Appendix D.5.

Baselines. 1) *Traditional VRP solvers:* we employ Concorde (Applegate et al., 2006) and LKH3 (Helsgaun, 2017) for solving TSP, and the hybrid genetic search (HGS) (Vidal, 2022) and LKH3 for CVRP. 2) *Neural methods:* we compare our method with POMO-based methods, including the original POMO (Kwon et al., 2020), AMDKD-POMO (Bi et al., 2022) and Meta-POMO (Manchanda et al., 2022) for TSP and CVRP. AMDKD-POMO is a recent method that improves the cross-distribution generalization of POMO using knowledge distillation. Meta-POMO uses Reptile to improve the generalization performance across both size and distribution. For a fair comparision, we re-train all methods following our training setups. Note that the setting of Meta-POMO is the most relevant to ours. As shown in Appendix D.2, we tune the key hyperparameters (i.e., β and K) of Meta-POMO since we empirically find its straightforward adaptation to POMO (i.e., *decaying* β in Table 4) performs poorly. We also show the results of their open-sourced pre-trained models (i.e., POMO* and AMDKD-POMO*) in Table 1, with the aim of demonstrating the severe generalization issue of current neural methods rather than the direct comparison. Specifically, POMO* is trained on instances with a fixed size and distribution (i.e., $n = 100$ with the uniform distribution), and AMDKD-POMO* is adaptively distilling from teacher models trained on fixed-sized instances following different distributions (i.e., $n = 100$ with the uniform, cluster and mixed distributions). More implementation details are provided in Appendix D.1.

Training Setups. We follow most of the setups in Kwon et al. (2020). For our method, Adam optimizer (Kingma & Ba, 2015) is used in both inner-loop and outer-loop optimization, with the weight decay of $1e-6$. The step sizes (learning rates) are $\alpha = \beta = 1e-4$, and decayed by 10 in the last 10% iterations to achieve a faster convergence. The batch size is $M = 64$ ($M = 32$ for instances with sizes larger than 150). The training task set consists of hundreds of (i.e., 341) tasks, with diverse sizes $\mathcal{N} = [50, 200]$ and distributions (i.e., uniform (U) and gaussian mixture (GM) distributions). More details about the generation of training and test data are presented in Appendix B. Similar to Finn et al. (2017), we simply set $B = K = 1$ and empirically observe strong performance. As suggested by Kwon et al. (2020), most of the training is already completed by 200 epochs (i.e., 20M instances) for POMO. We give more instances due to our complicated problem setting. Specifically, we re-train all methods for roughly the same number of instances (i.e., 32M) sampled from our training task set. For example, we re-train POMO for roughly 500K iterations (i.e., gradient updates). For our method, one iteration of meta-training consists of a pair of inner-loop and out-loop optimization, which needs two batches of instances. Therefore, for a fair comparison, we train our method for roughly 250K iterations. For the hierarchical task scheduler, we set $\eta = 1$ and $E_s = 225$K. It evaluates the hardness and updates the weight of each task every 100 iterations. Due to the training efficiency, we regard meta-training with the first-order approximation (i.e., *Ours*) as the default method, which uses the second-order derivatives in the first 50K iterations, and switch to the first-order ones afterwards. In

[3] https://github.com/yd-kwon/POMO

Table 1. Evaluation on cross-size or distribution generalization. (* marks results derived by open-sourced pretrained models.)

	Method	**Cross-Distribution Generalization** (1K ins.)						**Cross-Size Generalization** (1K ins.)					
		$(200, GM_2^5)$		$(200, R)$		$(200, E)$		$(300, U)$		$(300, GM_3^{10})$		$(300, GM_7^{50})$	
		Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time
TSP	Concorde	8.78 (0.00%)	0.6m	8.20 (0.00%)	0.5m	8.09 (0.00%)	0.5m	12.95 (0.00%)	1.4m	9.47 (0.00%)	1.2m	5.63 (0.00%)	1.0m
	LKH3	8.78 (0.00%)	3.3m	8.20 (0.00%)	3.3m	8.09 (0.00%)	3.5m	12.95 (0.00%)	5.9m	9.47 (0.00%)	12.5m	5.63 (0.01%)	18.5m
	POMO*	9.36 (6.67%)	0.5m	8.41 (2.66%)	0.5m	8.28 (2.35%)	0.5m	13.82 (6.70%)	1.5m	10.73 (13.38%)	1.5m	6.54 (16.04%)	1.5m
	AMDKD-POMO*	9.05 (2.97%)	0.5m	8.41 (2.57%)	0.5m	8.30 (2.61%)	0.5m	13.97 (7.83%)	1.5m	10.25 (8.22%)	1.5m	6.25 (11.00%)	1.5m
	POMO	**9.01 (2.56%)**	0.5m	8.37 (2.13%)	0.5m	8.24 (1.85%)	0.5m	13.54 (4.51%)	1.5m	9.88 (4.27%)	1.5m	5.83 (3.46%)	1.5m
	AMDKD-POMO	9.10 (3.56%)	0.5m	8.47 (3.32%)	0.5m	8.38 (3.55%)	0.5m	13.74 (6.08%)	1.5m	9.97 (5.30%)	1.5m	6.00 (6.44%)	1.5m
	Meta-POMO	9.03 (2.78%)	0.5m	8.39 (2.31%)	0.5m	8.25 (2.00%)	0.5m	13.50 (4.23%)	1.5m	9.89 (4.38%)	1.5m	5.80 (2.94%)	1.5m
	Ours-SO	9.01 (2.59%)	0.5m	**8.36 (1.99%)**	0.5m	**8.23 (1.72%)**	0.5m	**13.37 (3.22%)**	1.5m	**9.82 (3.72%)**	1.5m	**5.78 (2.61%)**	1.5m
	Ours	9.02 (2.71%)	0.5m	8.37 (2.14%)	0.5m	8.24 (1.86%)	0.5m	13.40 (3.42%)	1.5m	9.84 (3.89%)	1.5m	5.79 (2.75%)	1.5m
	Meta-POMO+FS ($K = 1$)	9.02 (2.74%)	2.0m	8.38 (2.24%)	2.0m	8.25 (1.92%)	2.0m	13.46 (3.87%)	6.8m	9.86 (4.11%)	6.8m	5.78 (2.64%)	6.8m
	Meta-POMO+FS ($K = 10$)	9.02 (2.67%)	15.7m	8.38 (2.17%)	15.7m	8.24 (1.83%)	15.7m	13.42 (3.58%)	0.9h	9.84 (3.91%)	0.9h	**5.77 (2.46%)**	0.9h
	Ours-SO+FS ($K = 1$)	**9.01 (2.53%)**	2.0m	**8.36 (1.95%)**	2.0m	**8.22 (1.63%)**	2.0m	**13.35 (3.05%)**	6.8m	**9.81 (3.57%)**	6.8m	**5.77 (2.47%)**	6.8m
	Ours+FS ($K = 1$)	9.01 (2.60%)	2.0m	8.37 (2.05%)	2.0m	8.23 (1.74%)	2.0m	13.37 (3.19%)	6.8m	9.82 (3.69%)	6.8m	5.78 (2.52%)	6.8m
CVRP	HGS	18.89 (0.00%)	0.7h	19.36 (0.00%)	0.6h	19.45 (0.00%)	0.5h	25.61 (0.00%)	1.0h	22.20 (0.00%)	1.6h	22.11 (0.00%)	0.9h
	LKH3	19.09 (1.06%)	0.6h	19.55 (0.99%)	0.6h	19.66 (1.04%)	0.5h	25.97 (1.39%)	0.6h	22.46 (1.19%)	0.7h	22.24 (0.59%)	0.6h
	POMO*	19.93 (5.60%)	0.6m	20.45 (5.74%)	0.6m	20.54 (5.69%)	0.6m	28.72 (12.32%)	1.8m	24.81 (12.00%)	1.8m	24.33 (10.22%)	1.8m
	AMDKD-POMO*	20.29 (7.59%)	0.6m	20.89 (8.07%)	0.6m	20.96 (7.92%)	0.6m	30.49 (19.17%)	1.8m	25.65 (15.92%)	1.8m	24.41 (10.60%)	1.8m
	POMO	19.47 (3.12%)	0.6m	19.99 (3.32%)	0.6m	20.12 (3.49%)	0.6m	27.07 (5.74%)	1.8m	23.25 (4.80%)	1.8m	22.80 (3.16%)	1.8m
	AMDKD-POMO	19.58 (3.69%)	0.6m	20.07 (3.72%)	0.6m	20.20 (3.93%)	0.6m	26.94 (5.20%)	1.8m	23.28 (4.92%)	1.8m	22.82 (3.27%)	1.8m
	Meta-POMO	19.48 (3.19%)	0.6m	20.01 (3.40%)	0.6m	20.15 (3.65%)	0.6m	26.87 (4.94%)	1.8m	23.09 (4.07%)	1.8m	22.75 (2.93%)	1.8m
	Ours-SO	**19.38 (2.66%)**	0.6m	**19.91 (2.87%)**	0.6m	**20.05 (3.13%)**	0.6m	26.67 (4.15%)	1.8m	**22.93 (3.33%)**	1.8m	**22.60 (2.23%)**	1.8m
	Ours	19.39 (2.69%)	0.6m	19.91 (2.88%)	0.6m	20.07 (3.21%)	0.6m	**26.65 (4.10%)**	1.8m	22.93 (3.35%)	1.8m	22.61 (2.27%)	1.8m
	Meta-POMO+FS ($K = 1$)	19.43 (2.92%)	2.4m	19.96 (3.13%)	2.4m	20.10 (3.39%)	2.4m	26.71 (4.32%)	8.2m	22.99 (3.64%)	8.2m	22.70 (2.72%)	8.2m
	Meta-POMO+FS ($K = 10$)	19.41 (2.83%)	18.8m	19.94 (3.03%)	18.8m	20.08 (3.27%)	18.8m	26.65 (4.07%)	1.1h	22.95 (3.43%)	1.1h	22.67 (2.55%)	1.1h
	Ours-SO+FS ($K = 1$)	**19.38 (2.65%)**	2.4m	**19.90 (2.81%)**	2.4m	**20.03 (3.00%)**	2.4m	26.61 (3.93%)	8.2m	**22.90 (3.21%)**	8.2m	**22.58 (2.16%)**	8.2m
	Ours+FS ($K = 1$)	19.38 (2.66%)	2.4m	19.90 (2.83%)	2.4m	20.04 (3.05%)	2.4m	**26.61 (3.92%)**	8.2m	22.91 (3.23%)	8.2m	22.59 (2.20%)	8.2m

specific, the meta-training with full second-order derivatives (i.e., *Ours-SO*) needs roughly 5 days and 53GB GPU memory for TSP (6 days and 71GB GPU memory for CVRP), while *Ours* needs 2.5 days and 17GB GPU memory for TSP (3 days and 25GB GPU memory for CVRP).

Inference Setups. For all neural methods, we use the greedy rollout with x8 instance augmentations following Kwon et al. (2020). We report the average results over the test dataset containing 1K instances. The reported time is the total time to solve the entire test dataset. The reported gaps are computed with respect to the traditional VRP solvers (i.e., Concorde for TSP, and HGS for CVRP). Specifically, we evaluate the effectiveness of our method on the *zero-shot* and *few-shot* (FS) settings. For the zero-shot setting, the trained model is directly used to construct the solutions. We further evaluate meta-learning based methods (i.e., Meta-POMO, Ours-SO and Ours) on the few-shot setting, where we fine-tune the meta-model for K iterations only using extra 1K instances sampled from the test task (0.003% of instances used for meta-training). The instances are augmented following Kwon et al. (2020). Note that the instances for fine-tuning are different from the test ones. The Adam optimizer is used with the learning rate of $\alpha = 1e-5$ and the weight decay of $1e-6$. Moreover, we further combine our method with EAS (Hottung et al., 2022) when evaluating on benchmark instances (see Appendix D.3).

5.1. Performance Evaluation

Below, we demonstrate the effectiveness of our method on synthetic and real-world datasets. For the synthetic data, we evaluate the generalization performance across size, distribution and the both. Note that we conduct t-test (with threshold of 5%) to verify the statistical significance, if the average objectives of two neural methods are close.

Cross-Size or Distribution Generalization. We first consider a simple setting where either the cross-size or distribution generalization is evaluated. For the cross-distribution setting, we test on instances of size $n = 200 \in [50, 200]$, while following diverse distributions that are unseen during training. Note that we do not strictly choose the test tasks sampled from the presumed training task distribution $p(\mathcal{T})$ since it only covers a small part of the entire problem space. Therefore, we also evaluate all methods on several complex distributions, e.g., rotation (R) and explosion (E) distributions (Bossek et al., 2019). For the cross-size setting, we evaluate on instances of the size $n = 300 \notin [50, 200]$ following distributions used in training. Besides the zero-shot setting, we further compare with another meta-learning based method (i.e., Meta-POMO) on the few-shot setting, where we fine-tune the learned model for K steps only using limited data. The detailed results are shown in Table 1, where we observe that our method can achieve superior performance on both settings. The inferior performance

Table 2. Evaluation on cross-size and distribution generalization.

	Method	Cross-Size and Distribution Generalization (1K ins.)											
		$(300, R)$		$(300, E)$		$(500, R)$		$(500, E)$		$(1000, R)$		$(1000, E)$	
		Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time	Obj. (Gap)	Time
TSP	Concorde	9.79 (0.00%)	1.2m	9.48 (0.00%)	1.5m	12.39 (0.00%)	5.0m	11.73 (0.00%)	5.8m	17.09 (0.00%)	0.7h	15.66 (0.00%)	0.9h
	LKH3	9.79 (0.00%)	6.0m	9.48 (0.00%)	6.8m	12.39 (0.00%)	11.8m	11.73 (0.00%)	13.8m	17.09 (0.00%)	0.4h	15.66 (0.00%)	0.5h
	POMO	10.23 (4.43%)	1.5m	9.88 (4.20%)	1.5m	13.63 (10.00%)	6.0m	12.89 (9.88%)	6.0m	20.74 (21.38%)	0.8h	18.94 (20.97%)	0.8h
	AMDKD-POMO	10.35 (5.69%)	1.5m	10.06 (6.15%)	1.5m	13.74 (10.85%)	6.0m	13.08 (11.52%)	6.0m	20.73 (21.25%)	0.8h	19.08 (21.85%)	0.8h
	Meta-POMO	10.22 (4.37%)	1.5m	9.87 (4.14%)	1.5m	13.56 (9.41%)	6.0m	12.84 (9.44%)	6.0m	20.51 (19.97%)	0.8h	18.77 (19.88%)	0.8h
	Ours-SO	**10.14 (3.54%)**	1.5m	**9.78 (3.13%)**	1.5m	**13.39 (8.07%)**	6.0m	**12.64 (7.73%)**	6.0m	**20.37 (19.20%)**	0.8h	**18.59 (18.74%)**	0.8h
	Ours	10.16 (3.74%)	1.5m	9.80 (3.35%)	1.5m	13.42 (8.30%)	6.0m	12.66 (7.90%)	6.0m	20.40 (19.36%)	0.8h	18.60 (18.80%)	0.8h
	Meta-POMO+FS ($K = 1$)	10.18 (3.96%)	6.8m	9.83 (3.70%)	6.8m	13.34 (7.60%)	0.5h	12.63 (7.66%)	0.5h	19.58 (14.52%)	6.5h	17.92 (14.48%)	6.5h
	Meta-POMO+FS ($K = 10$)	10.16 (3.69%)	0.9h	9.80 (3.41%)	0.9h	13.23 (6.75%)	4.1h	12.54 (6.84%)	4.1h	–	–	–	–
	Ours-SO+FS ($K = 1$)	**10.12 (3.32%)**	6.8m	**9.76 (2.91%)**	6.8m	**13.19 (6.45%)**	0.5h	**12.45 (6.11%)**	0.5h	**19.53 (14.28%)**	6.5h	17.79 (13.65%)	6.5h
	Ours+FS ($K = 1$)	10.13 (3.41%)	6.8m	9.77 (3.05%)	6.8m	13.20 (6.52%)	0.5h	12.51 (6.64%)	0.5h	19.53 (14.30%)	6.5h	**17.75 (13.38%)**	6.5h
CVRP	HGS	22.40 (0.00%)	1.3h	23.02 (0.00%)	1.3h	26.62 (0.00%)	4.5h	26.89 (0.00%)	4.6h	32.36 (0.00%)	30.9h	32.01 (0.00%)	37.7h
	LKH3	22.68 (1.28%)	0.7h	23.32 (1.28%)	0.7h	27.06 (1.69%)	0.9h	27.32 (1.61%)	0.9h	33.16 (2.51%)	1.6h	32.78 (2.43%)	1.6h
	POMO	23.56 (5.30%)	1.8m	24.20 (5.30%)	1.8m	29.06 (9.48%)	6.9m	29.29 (9.29%)	6.9m	39.33 (22.44%)	1.0h	38.63 (21.73%)	1.0h
	AMDKD-POMO	23.54 (5.18%)	1.8m	24.24 (5.39%)	1.8m	29.06 (9.32%)	6.9m	29.33 (9.29%)	6.9m	39.72 (23.17%)	1.0h	38.86 (21.90%)	1.0h
	Meta-POMO	23.39 (4.54%)	1.8m	24.08 (4.71%)	1.8m	28.53 (7.34%)	6.9m	28.80 (7.32%)	6.9m	37.46 (16.09%)	0.9h	36.85 (15.52%)	0.9h
	Ours-SO	23.24 (3.83%)	1.8m	**23.93 (4.07%)**	1.8m	28.34 (6.60%)	6.7m	28.63 (6.69%)	6.7m	37.30 (15.62%)	0.8h	36.61 (14.83%)	0.8h
	Ours	**23.23 (3.79%)**	1.8m	23.94 (4.08%)	1.8m	**28.29 (6.41%)**	6.7m	**28.60 (6.56%)**	6.7m	**37.02 (14.73%)**	0.8h	**36.40 (14.15%)**	0.8h
	Meta-POMO+FS ($K = 1$)	23.29 (4.05%)	8.2m	23.96 (4.20%)	8.2m	28.13 (5.80%)	0.6h	28.43 (5.90%)	0.6h	36.14 (11.93%)	7.5h	35.78 (12.07%)	7.5h
	Meta-POMO+FS ($K = 10$)	23.23 (3.79%)	1.1h	23.90 (3.92%)	1.1h	**27.95 (5.14%)**	4.9h	**28.24 (5.19%)**	4.7h	–	–	–	–
	Ours-SO+FS ($K = 1$)	23.19 (3.61%)	8.2m	**23.87 (3.78%)**	8.2m	28.03 (5.41%)	0.6h	28.33 (5.52%)	0.6h	35.69 (10.52%)	7.4h	35.40 (10.92%)	7.4h
	Ours+FS ($K = 1$)	**23.19 (3.59%)**	8.2m	23.87 (3.79%)	8.2m	**28.01 (5.34%)**	0.6h	**28.31 (5.44%)**	0.6h	**35.60 (10.26%)**	7.4h	**35.25 (10.45%)**	7.4h

of AMDKD-POMO may be attributed to its design for the trivial problem setting and sample inefficiency. While it is specialized for the cross-distribution generalization, its original problem setting is much easier than ours, with only three distributions on the fixed size (i.e., 100) considered during training. To achieve satisfactory performance, a good pretrained model for each training task is needed, which requires a huge amount of training instances.

Cross-Size and Distribution Generalization. We further evaluate all methods on a much more complex setting, where the generalization across both size and distribution is considered. Specifically, we choose the test task with the unseen size $n \in [300, 500, 1000]$ and distribution $d \in [R, E]$ during training. The results are presented in Table 2, where we observe our method has consistently better performance than baselines. Notably, our method achieves superior results on the large-scale CVRP1000 task with totally unseen distributions, showing a strong omni-generalization capability.

Results on Benchmark Datasets. We further evaluate all methods on the well-konwn benchmark datasets TSPLIB (Reinelt, 1991) and CVRPLIB (Set-X (Uchoa et al., 2017) and Set-XML100 (Queiroga et al., 2022)). Detailed results can be found in Appendix D.3, where we observe our method performs well in most cases.

5.2. Analyses

In this section, we conduct further analyses, including the ablation studies and few-shot experiments, to demonstrate the effectiveness and sensitivity of the proposed framework. More ablation studies on hyperparameters, optimizers and normalization layers are presented in Appendix D.4.

Table 3. Ablation study on Components.

	$(200, GM_2^5)$	$(300, U)$	$(500, R)$	$(1000, E)$
POMO	3.12%	5.74%	9.48%	21.73%
+ task scheduler	**2.67%**	4.44%	7.53%	19.03%
+ meta-training	3.08%	4.79%	7.02%	15.39%
Ours	2.69%	**4.10%**	**6.41%**	**14.15%**

Ablation Study on Components. In Section 5.1, we have shown the effect of the first-order approximation. Compared with the full second-order method, it could achieve similar or even better zero-shot and few-shot performance, and meanwhile greatly reduce the training complexity. Here, following the training setups presented in Section 5, we further conduct the ablation study on CVRP to demonstrate the benefit of each component in our framework. The results are shown in Table 3, where we observe that the meta-training significantly improves POMO (zero-shot) performance on the large-scale instances, and the task scheduler can further boost the overall performance of the meta-training.

Efficient Adaptation. As shown in Table 1 and 2, given the same amount of training instances, our method achieves strong zero-shot performance, which enables efficient adaptation to a new task afterwards. To demonstrate it, we further conduct two experiments on the adaptation to CVRP (500, R) only using 100 and 1000 instances. As shown

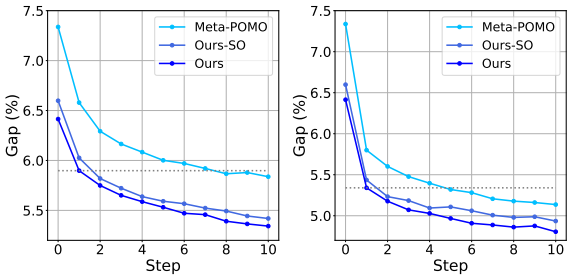


Figure 3. Adaptation to CVRP (500, R) test task using (a) 100 instances; (b) 1000 instances.

in Figure 3, our methods can be efficiently adapted to the new task, while Meta-POMO needs to run multiple steps to achieve similar few-shot performance (e.g., the dotted line for $K = 1$) to our method. Moreover, we observe that the number of instances is crucial to the few-shot performance in the reinforcement learning. Meta-POMO may need more instances in order to achieve strong few-shot performance.

5.3. Generalizability

To evaluate the generalizability of the proposed framework, we further apply it to L2D (Li et al., 2021), which is an improvement-based method outperforming LKH3 on large-scale CVRP instances. Specifically, it decomposes the large-scale problem instance into several subproblems, which are selected by a (supervised) learned policy, and uses an existing solver (e.g., LKH3 or HGS) to solve each subproblem. We train the model on the omni-generalization setting, where the training task set consists of various sizes and distributions. The results show that our method could improve the generalization of L2D, demonstrating the effectiveness and generalizability of our method. The detailed training setups and empirical results are shown in Appendix D.5.

6. Conclusion

This paper studies the omni-generalization issue of neural methods across both problem size and distribution in VRPs. We propose a generic meta-learning framework to tackle this issue, which is model-agnostic and compatible with any model trained with gradient updates. We further provide analyses of the first-order approximation methods on the reinforcement learning setting, and propose a simple yet efficient method to reduce the meta-training complexity.

The limitations of this work are the training efficiency and scalability. However, they heavily depend on the base model and meta-learning algorithm. If a pretrained model exists, it would be better to conduct meta-training on it. We refer to Appendix E for further discussions. We leave advanced algorithms and other neural VRP methods to the future work. We hope our work could provide new insights for learning a more generalizable neural VRP heuristic in the community.

Acknowledgements

Wen Song was supported by the National Natural Science Foundation of China under Grant 62102228, and the Natural Science Foundation of Shandong Province under Grant ZR2021QF063. We would like to thank the anonymous reviewers and (S)ACs of ICML 2023 for their constructive comments and dedicated service to the community. Jianan Zhou would like to personally express deep gratitude to his grandmother, Zhiling Kang, for her meticulous care and love during last 25 years. Eternal easy rest in sweet slumber.

References

Ahn, S., Seo, Y., and Shin, J. Learning what to defer for maximum independent sets. In *ICML*, pp. 134–144. PMLR, 2020.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. Concorde tsp solver. 2006.

Augerat, P. *Approche polyèdrale du problème de tournées de véhicules*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1995.

Bachlechner, T., Majumder, B. P., Mao, H., Cottrell, G., and McAuley, J. Rezero is all you need: Fast convergence at large depth. In *UAI*, pp. 1352–1361. PMLR, 2021.

Bdeir, A., Falkner, J. K., and Schmidt-Thieme, L. Attention, filling in the gaps for generalization in routing problems. In *ECMLPKDD*, 2022.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. In *ICLR Workshop Track*, 2017.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *ICML*, pp. 41–48, 2009.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Bi, J., Ma, Y., Wang, J., Cao, Z., Chen, J., Sun, Y., and Chee, Y. M. Learning generalizable models for vehicle routing problems via knowledge distillation. In *NeurIPS*, 2022.

Bossek, J., Kerschke, P., Neumann, A., Wagner, M., Neumann, F., and Trautmann, H. 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*, pp. 58–71, 2019.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Cattaruzza, D., Absi, N., Feillet, D., and González-Feliu, J. Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, 6(1):51–79, 2017.

Chen, X. and Tian, Y. Learning to perform local rewriting for combinatorial optimization. In *NeurIPS*, volume 32, 2019.

Croes, G. A. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

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

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, volume 30, 2017.

Drakulic, D., Michel, S., Mai, F., Sors, A., and Andreoli, J.-M. Bq-nco: Bisimulation quotienting for generalizable neural combinatorial optimization. *arXiv preprint arXiv:2301.03313*, 2023.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pp. 1126–1135. PMLR, 2017.

Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D., and Singh, S. Bootstrapped meta-learning. In *ICLR*, 2022.

Fu, Z.-H., Qiu, K.-B., and Zha, H. Generalize a small pre-trained model to arbitrarily large tsp instances. In *AAAI*, volume 35, pp. 7474–7482, 2021.

Garaix, T., Artigues, C., Feillet, D., and Josselin, D. Vehicle routing problems with alternative paths: An application to on-demand transportation. *European Journal of Operational Research*, 204(1):62–75, 2010.

Geisler, S., Sommer, J., Schuchardt, J., Bojchevski, A., and Günnemann, S. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *ICLR*, 2022.

Helsgaun, K. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

Helsgaun, K. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, pp. 24–50, 2017.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

Hottung, A. and Tierney, K. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence*, pp. 443–450. IOS Press, 2020.

Hottung, A., Kwon, Y.-D., and Tierney, K. Efficient active search for combinatorial optimization problems. In *ICLR*, 2022.

Hou, Q., Yang, J., Su, Y., Wang, X., and Deng, Y. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *ICLR*, 2023.

Hudson, B., Li, Q., Malencia, M., and Prorok, A. Graph neural network guided local search for the traveling salesperson problem. In *ICLR*, 2022.

Jiang, Y., Wu, Y., Cao, Z., and Zhang, J. Learning to solve routing problems via distributionally robust optimization. In *AAAI*, 2022.

Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., and Laurent, T. Learning tsp requires rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming*, 2021.

Kim, M., Park, J., et al. Learning collaborative policies to solve np-hard routing problems. In *NeurIPS*, volume 34, pp. 10418–10430, 2021.

Kim, M., Park, J., and Park, J. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *NeurIPS*, 2022a.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Konstantakopoulos, G. D., Gayialis, S. P., and Kechagias, E. P. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational research*, pp. 1–30, 2022.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *ICLR*, 2018.

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., and Min, S. Pomo: Policy optimization with multiple optima for reinforcement learning. In *NeurIPS*, volume 33, pp. 21188–21198, 2020.

Kwon, Y.-D., Choo, J., Yoon, I., Park, M., Park, D., and Gwon, Y. Matrix encoding networks for neural combinatorial optimization. In *NeurIPS*, volume 34, pp. 5138–5149, 2021.

Li, S., Yan, Z., and Wu, C. Learning to delegate for large-scale vehicle routing. In *NeurIPS*, volume 34, pp. 26198–26211, 2021.

Lisicki, M., Afkanpour, A., and Taylor, G. W. Evaluating curriculum learning strategies in neural combinatorial optimization. In *NeurIPS 2020 Workshop on Learning Meets Combinatorial Algorithms*, 2020.

Liu, S., Zhang, Y., Tang, K., and Yao, X. How good is neural combinatorial optimization? *arXiv preprint arXiv:2209.10913*, 2022.

Lu, H., Zhang, X., and Yang, S. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2020.

Ma, Q., Ge, S., He, D., Thaker, D., and Drori, I. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.

Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., and Tang, J. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *NeurIPS*, volume 34, pp. 11096–11107, 2021.

Manchanda, S., Michel, S., Drakulic, D., and Andreoli, J.-M. On the generalization of neural combinatorial optimization heuristics. In *ECMLPKDD*, 2022.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS*, volume 31, 2018.

Nichol, A., Achiam, J., and Schulman, J. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Qiu, R., Sun, Z., and Yang, Y. DIMES: A differentiable meta solver for combinatorial optimization problems. In *NeurIPS*, 2022.

Queiroga, E., Sadykov, R., Uchoa, E., and Vidal, T. 10,000 optimal cvrp solutions for testing machine learning based heuristics. In *AAAI Workshop on Machine Learning for Operations Research (ML4OR)*, 2022.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *ICLR*, 2020.

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

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *ICLR*, 2020.

Selsam, D., Lamm, M., Benedikt, B., Liang, P., de Moura, L., Dill, D. L., et al. Learning a sat solver from single-bit supervision. In *ICLR*, 2019.

Shaw, P. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pp. 417–431. Springer, 1998.

Smith-Miles, K., Hemert, J. v., and Lim, X. Y. Understanding tsp difficulty by learning from evolved instances. In *International conference on learning and intelligent optimization*, pp. 266–280. Springer, 2010.

Sun, Z. and Yang, Y. Difusco: Graph-based diffusion solvers for combinatorial optimization. *arXiv preprint arXiv:2302.08224*, 2023.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, volume 30, 2017.

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

Vilalta, R. and Drissi, Y. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NeurIPS*, volume 28, 2015.

Wang, C., Yang, Y., Slumbers, O., Han, C., Guo, T., Zhang, H., and Wang, J. A game-theoretic approach for improving generalization ability of tsp solvers. In *ICLR 2022 Workshop on Gamification and Multiagent Solutions*, 2022.

Wang, R., Hua, Z., Liu, G., Zhang, J., Yan, J., Qi, F., Yang, S., Zhou, J., and Yang, X. A bi-level framework for learning to solve combinatorial optimization on graphs. In *NeurIPS*, volume 34, pp. 21453–21466, 2021.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Wolpert, D. H. and Macready, W. G. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Wu, Y., Song, W., Cao, Z., Zhang, J., and Lim, A. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 2021.

Wu, Y., Zhou, J., Xia, Y., Zhang, X., Cao, Z., and Zhang, J. Neural airport ground handling. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

Xin, L., Song, W., Cao, Z., and Zhang, J. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *AAAI*, volume 35, pp. 12042–12049, 2021a.

Xin, L., Song, W., Cao, Z., and Zhang, J. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. In *NeurIPS*, volume 34, pp. 7472–7483, 2021b.

Yehuda, G., Gabel, M., and Schuster, A. It's not what machines can learn, it's what we cannot teach. In *ICML*, pp. 10831–10841. PMLR, 2020.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., and Chi, X. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *NeurIPS*, volume 33, pp. 1621–1632, 2020.

Zhang, Z., Zhang, Z., Wang, X., and Zhu, W. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *AAAI*, 2022.

Zhou, J., Wu, Y., Cao, Z., Song, W., Zhang, J., and Chen, Z. Learning large neighborhood search for vehicle routing in airport ground handling. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

A. Analysis of First-Order Approximation

Without loss of generality, we consider optimization with the stochastic gradient descent (SGD), and treat each task equally. Therefore, the meta-model update in Reptile (Nichol et al., 2018) (i.e., Eq. (9)) could be rewritten as:

$$\theta_0 \leftarrow \theta_0 + \beta \frac{1}{B} \sum_{i=1}^{B} (\theta_i^{(K)} - \theta_0). \tag{10}$$

For each task $\mathcal{T}_i$, $(\theta_0 - \theta_i^{(K)})/\alpha$ could be viewed as the (Reptile) gradient term g_R in the SGD formulation, and α, β are the step sizes of inner-loop and outer-loop optimization, respectively. When $K = 1$, Reptile is equivalent to the joint training on the expected loss of the training tasks:

$$g_R^1 = \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})}[\frac{\theta_0 - \theta_i^{(1)}}{\alpha}] = \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})}[\frac{\theta_0 - (\theta_0 - \alpha \nabla_{\theta_0} \mathcal{L}_i(\theta_0))}{\alpha}] = \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})}[\nabla_{\theta_0} \mathcal{L}_i(\theta_0)]. \tag{11}$$

However, in this case, it is equivalent to the naive pretraining on a large training task set, which requires ad-hoc tricks to achieve desirable fine-tuning performance. When performing multiple gradient updates ($K > 1$) in the inner-loop optimization, Reptile is able to incorporate information from higher-order derivatives of the loss function. For the simplicity of notations, we omit the index for task i, and use the following definitions:

$$g^{(k)} = \frac{\partial \mathcal{L}(\theta^{(k)})}{\partial \theta^{(k)}}; \quad \tilde{g}^{(k)} = \frac{\partial \mathcal{L}(\theta^{(k)})}{\partial \theta^{(0)}}; \quad \tilde{h}^{(k)} = \frac{\partial \mathcal{L}^2(\theta^{(k)})}{\partial (\theta^{(0)})^2}; \quad k \in [0, K], \tag{12}$$

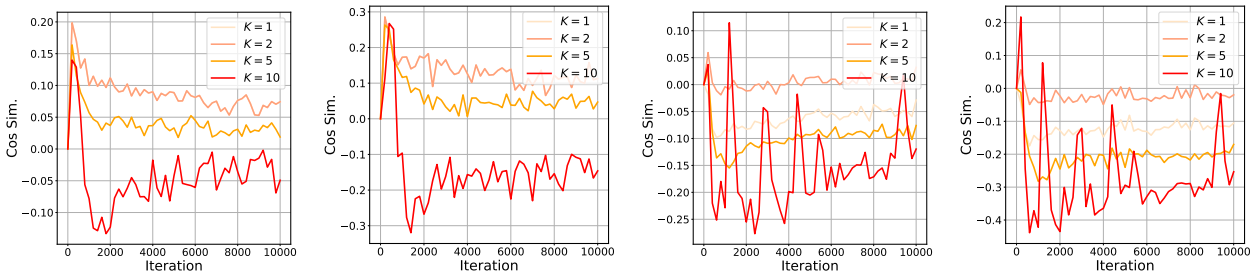
where $g^{(k)}, \tilde{g}^{(k)}$ are the gradients of the loss function with respect to (w.r.t.) the task-specific model $\theta^{(k)}$ and meta-model $\theta^{(0)} = \theta_0$, and $\tilde{h}^{(k)}$ is the hessian w.r.t. the meta-model. With the Taylor expansion, the gradient of the loss function w.r.t the task-specific model can be expressed as:

$$\begin{aligned} g^{(k)} = \frac{d\mathcal{L}}{d\theta}|_{\theta=\theta^{(k)}} &\approx \frac{d\mathcal{L}}{d\theta}|_{\theta=\theta^{(0)}} + \frac{d^2\mathcal{L}}{d\theta^2}|_{\theta=\theta^{(0)}} (\theta^{(k)} - \theta^{(0)}) \\ &\approx \frac{\partial \mathcal{L}(\theta^{(k)})}{\partial \theta^{(0)}} + \frac{\partial \mathcal{L}^2(\theta^{(k)})}{\partial (\theta^{(0)})^2} (\theta^{(0)} - \alpha \sum_{j=0}^{k-1} \frac{\partial \mathcal{L}(\theta^{(j)})}{\partial \theta^{(j)}} - \theta^{(0)}) \\ &\approx \tilde{g}^{(k)} - \alpha \tilde{h}^{(k)} \sum_{j=0}^{k-1} g^{(j)}. \end{aligned} \tag{13}$$

Indeed, FOMAML (Finn et al., 2017) simply drops the higher-order term and uses $g^{(K)}$ as the approximation to the second-order derivative, while Reptile approximates it in the following way:

$$g_R^K = \frac{1}{\alpha}(\theta_0 - \theta^{(K)}) = \frac{1}{\alpha}(\theta_0 - (\theta_0 - \alpha \sum_{j=0}^{K-1} g^{(j)})) = \sum_{j=0}^{K-1} g^{(j)}. \tag{14}$$

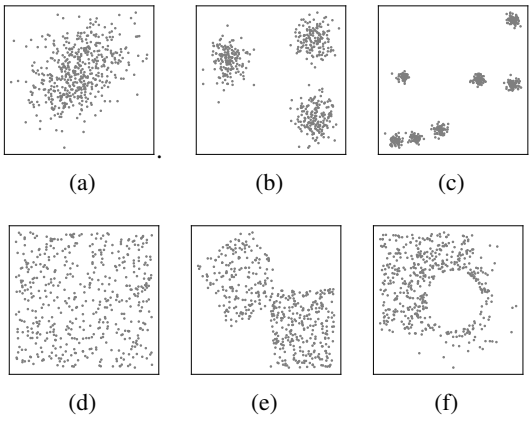
For example, if we run $K = 2$ steps in the inner-loop optimization, based on Eqs. (13)-(14), the gradient of Reptile is $g_R^2 = g^{(0)} + g^{(1)} \approx \tilde{g}^{(0)} + \tilde{g}^{(1)} - \alpha \tilde{h}^{(1)} g^{(0)}$, and the gradient of FOMAML is $g_F^2 = g^{(2)} \approx \tilde{g}^{(2)} - \alpha \tilde{h}^{(2)} (g^{(0)} + g^{(1)})$. However, it is non-trivial to execute the above derivations on our more complicated reinforcement learning (RL) setting (with Adam optimizer). Therefore, empirically, we further conduct an experiment to check whether g_R^K could serve as a good approximation on our setting. Specifically, similar to the setups presented in Appendix D.1, we meta-train POMO with Ours-SO for $K \in [1, 2, 5, 10]$ steps in the inner-loop optimization. We collect the gradient direction of the second-order derivative $\text{sign}(\tilde{g}^{(K)})$ and that of the Reptile's approximation $\text{sign}(g_R^K)$, and compute their cosine similarity. Moreover, since we use the Adam optimizer, we also try to load the gradient statistics (e.g., momentum in the optimizer for outer-loop optimization) when conducting the inner-loop optimization. As indicated in Figure 4, Reptile fails to well approximate the second-order derivatives on our RL setting. As the step K increases, the cosine similarity decreases accordingly, which may be attributed to the accumulated effect throughout the K steps. However, a larger step K empirically results in a relatively better zero-shot generalization performance (given the same amount of training instances) as shown in Appendix D.2.



(a) First layer (b) Last layer (c) First layer (d) Last layer

Figure 4. (a)-(b): the cosine similarity of gradient directions between the second-order derivative and the Reptile's approximation when not loading gradient statistics. Note that the results of $K = 1$ and $K = 2$ are almost the same; (c)-(d): the cosine similarity of gradient directions between the second-order derivative and the Reptile's approximation when loading gradient statistics.

B. Data Generation



(a) (b) (c) (d) (e) (f)

Figure 5. TSP500 instances following various distributions: (a) GM_2^5: Gaussian mixture distribution with $c = 2, l = 5$; (b) GM_3^{10}: Gaussian mixture distribution with $c = 3, l = 10$; (c) GM_7^{50}: Gaussian mixture distribution with $c = 7, l = 50$; (d) U: Uniform distribution; (e) R: Rotation distribution; (f) E: Explosion distribution.

We consider VRP instances with various sizes and distributions. Since generating instances with different sizes is relatively easy, here we focus on the details regarding the generation of different distributions of node coordinates. To provide diverse data for meta-training, we generate instances following the uniform distribution, and the gaussian mixture distribution, which is demonstrated to be effective in capturing different hardness levels of the instances (Smith-Miles et al., 2010). We further generate instances following rotation and explosion distributions[4] to evaluate the generalization of the learned model. Below, we provide the details on the data generation procedure.

Uniform distribution. Following the convention (Kool et al., 2018; Kwon et al., 2020), as shown in Figure 5(d), the node coordinate of each node is uniformly sampled from the unit square $U(0, 1)$.

Gaussian mixture distribution. Following Zhang et al. (2022); Manchanda et al. (2022), we parameterize the gaussian mixture distribution with two hyperparameters, cluster c and scale l. For the simplicity of notations, we denote it as GM_c^l. Specifically, we first generate the coordinate of the center node v_{c_i} of each cluster c_i by uniformly sampling from $U(0, l)$. Other nodes $\mathcal{V} \setminus \{v_{c_i}\}_{i=1}^c$ are then equally distributed into c clusters, where the coordinates of nodes in each cluster forms a (multivariate) gaussian distribution. For example, if a node belongs to the cluster c_i, its coordinate is sampled from $\mathcal{N}(n_{c_i}, \mathbf{I})$, where the mean is the coordinate of the center node, and the covariance matrix is the identity matrix. Finally, we scale and translate the range of coordinates into $[0, 1]$ using the min-max normalization. Some exemplary instances are shown in Figure 5(a)-5(c).

Rotation distribution. Following Bossek et al. (2019), we mutate nodes, which originally follow the uniform distribution, by rotating a subset of them (anchored in the origin of the Euclidean plane) as shown in Figure 5(e). The coordinates of selected nodes are transformed by multiplying with $\begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix}$, where $\varphi \sim [0, 2\pi]$ is the rotation angle.

[4] https://github.com/jakobbossek/tspgen

Explosion distribution. Following Bossek et al. (2019), we mutate nodes, which originally follow the uniform distribution, by simulating a random explosion. Specifically, we first randomly select the center of the explosion v_c (i.e., the hole in Figure 5(f)). All nodes v_i within the explosion radius $R = 0.3$ is moved away from the center with the transformation form of $v_i = v_c + (R + s) \cdot \frac{v_c - v_i}{||v_c - v_i||}$, where $s \sim \text{Exp}(\lambda = 1/10)$ is a random value drawn from an exponential distribution.

In this paper, we mainly consider the distribution of node coordinates. For CVRP instances, following Kool et al. (2018); Kwon et al. (2020), the coordinate of the depot node v_0 is uniformly sampled from the unit square $U(0, 1)$. The demand of each node δ_i is randomly sampled from a discrete uniform distribution $\{1, \cdots, 9\}$. The capacity of each vehicle is set to $Q = \lceil 30 + \frac{n}{5} \rceil$, where $n \geq 50$ is the size of CVRP instances. The demand and capacity are further normalized to $\delta'_i = \delta/Q$ and 1, respectively. During meta-training, we generate the diverse task set with hundreds of tasks $\mathcal{T}(n, d)$, where $n \in \mathcal{N} = \{50, 55, \cdots, 200\}$ and $d(c, l) \in \mathcal{D} = \{(0, 0), (1, 1)\} \cup \{c[3, 5, 7] \times l[10, 30, 50]\}$. We denote the uniform distribution as $d(c = 0, l = 0)$. For each size, there are 11 tasks with different distributions in the training task set, and therefore $31 \times 11 = 341$ tasks in total.

C. POMO

POMO (Kwon et al., 2020) significantly improves upon AM (Kool et al., 2018) by exploiting the symmetry property, which inherently exists in the VRP solution. For example, a solution to a TSP instance is represented as a sequence of nodes. Multiple representations (with different start nodes) exist for the same solution. Previous method (Kool et al., 2018) selects the start node by the model (using a trainable START token from the NLP community). Since the solution construction process is formulated as a MDP, the first action (i.e., start node) may considerably affect the following actions. However, a desirable model should always be able to construct the optimal solution given different start nodes. POMO considers this symmetry property into the objective function, and the estimated gradient (i.e., Eq. (3)) could be rewritten as:

$$\nabla_\theta \mathcal{L}(\theta|\mathcal{G}) \approx \frac{1}{S} \sum_{s=1}^{S} (c(\tau_s) - b(\mathcal{G})) \nabla_\theta \log p_\theta(\tau_s|\mathcal{G}), \tag{15}$$

where τ_s is the solution with start node $v_s \in \mathcal{V}$, S is the number of start nodes (e.g., the size of an instance in TSP), and $b(\mathcal{G}) = \frac{1}{S} \sum_{s=1}^{S} c(\tau_s)$. Intuitively, Eq. (15) forces diverse trajectories towards optimal solution(s). Besides, POMO also leverages instance augmentations to enhance the inference performance. A brief summary is presented in Algorithm 2.

Algorithm 2 POMO with REINFORCE

Input: training instances $\{\mathcal{G}_m\}_{m=1}^{M}$, number of start nodes S, model θ;
Output: estimated gradient $\nabla_\theta \mathcal{L}(\theta)$;

1: $\{v_1^m, v_2^m, \cdots, v_S^m\} \leftarrow$ SelectStartingNodes($\mathcal{G}_m$) $\quad \forall m \in \{1, \cdots, M\}$
2: $\tau_s^m \leftarrow \theta(\mathcal{G}_m, v_s^m) \quad \forall m \in \{1, \cdots, M\}, \forall s \in \{1, \cdots, S\}$
3: $b(\mathcal{G}_m) \leftarrow \frac{1}{S} \sum_{s=1}^{S} c(\tau_s^m) \quad \forall m \in \{1, \cdots, M\}$
4: $\nabla_\theta \mathcal{L}(\theta) \leftarrow \frac{1}{MS} \sum_{m=1}^{M} \sum_{s=1}^{S} (c(\tau_s^m) - b(\mathcal{G}_m)) \nabla_\theta \log p_\theta(\tau_s^m|\mathcal{G}_m)$

D. Experiments

D.1. Extra Setups

Setups for Experiments in Figure 2. We follow the training setups presented in Section 5. For the left panel of Figure 2, we show the validation result of each method on CVRP (300, R). Specifically, *Ours-SO* refers to the second-order method presented in Algorithm 1, *FOMAML* refers to its first-order approximation, where we replace the line 16 of Algorithm 1 with Eq. (8), and *Ours* refers to the proposed approximation method, where we simply early-stop using the second-order derivative when the training tends to be stable (i.e., at the 50K_{th} iteration), and leverage the first-order one afterwards. For the right panel of Figure 2, the training process follows *Ours-SO*. However, in each iteration of meta-training, besides calculating the direction of the second-order derivative $\text{sign}(\nabla_{\theta_0} \mathcal{L}_i(\theta_i^{(K)}))$, we also collect that of FOMAML $\text{sign}(\nabla_{\theta_i} \mathcal{L}_i(\theta_i^{(K)}))$ and Reptile $\text{sign}(\theta_0 - \theta_i^{(K)})$ using the same batch of instances. Note that these two gradients are not used for meta updates. Then, we compute the cosine similarities of these gradient directions with the second-order one (i.e., $\text{sign}(\nabla_{\theta_0} \mathcal{L}_i(\theta_i^{(K)}))$), and show the average result over 500 iterations in the right panel of Figure 2.

Training Setups for Baselines. We conduct all experiments on a machine with NVIDIA A100 PCIe (80GB) cards and AMD EPYC 7513 CPU at 2.6GHz. As shown in Section 5, we compare our method with several strong baselines. Following the conventional setups in the community (Kool et al., 2018; Kwon et al., 2020; Hottung et al., 2022), for traditional VRP solvers such as Concorde, LKH3 and HGS, we run them on 32 CPU cores for solving TSP and CVRP instances, while running neural VRP methods on one GPU card. Below, we provide implementation details of all baselines. 1) Concorde (Applegate et al., 2006): We use Concorde[5] Version 03.12.19 with the default setting, to solve TSP instances. 2) LKH3 (Helsgaun, 2017): We use LKH3[6] Version 3.0.7 to solve TSP and CVRP instances. For each instance, we run LKH3 with 10000 trails and 10 runs. 3) HGS (Vidal, 2022): We run HGS[7] with the default hyperparameters to solve CVRP instances. The maximum number of iterations (without improvement) is set to 20000. 4) For POMO (Kwon et al., 2020), following the training setups presented in Section 5, we re-train it for 500K iterations with totally 32M instances, which are randomly sampled from our training task set. 5) AMDKD-POMO (Bi et al., 2022) tackles the cross-distribution generalization of neural methods using knowledge distillation. Specifically, it leverages the knowledge from multiple teacher models pretrained on different distributions to yield a generalizable student model. However, it is computationally intractable to obtain a pretrained model for each task since we have hundreds of training tasks on our problem setting. Therefore, following the default setting of Bi et al. (2022), we pretrain three teacher models on instances of size $n = 200$, but with distributions chosen from our training task set (i.e., the uniform U and gaussian mixture distributions GM_3^{10}, GM_7^{50}). We train each teacher model using 6.4M instances. After pretraining, we train a light-weight yet generalizable student model by adaptively distilling from the teacher models on another set of 12.8M instances ($n = 200$), so that the total amount of training instances (i.e., 32M) is close to other methods. 6) Meta-POMO (Manchanda et al., 2022) leverages Reptile (Nichol et al., 2018), which does not need to split data into training and validation sets. Therefore, based on its default setting and our experiments in Appendix D.2, we set $\beta = 0.9, B = 1, K = 50$ and meta-train POMO using Reptile for 10K iterations to keep the same amount of training instances as other methods.

D.2. Tuning on Meta-POMO

As shown in Table 4, we empirically observe the inferior performance of Meta-POMO with a decaying step size β, which is the straightforward adaptation of Manchanda et al. (2022) to POMO. Specifically, the step size is gradually decayed with the form of $\beta_t = \beta_0 \times \gamma^t$, where $\beta_0 = 0.99$ is the initial step size, $\gamma = 0.999$ is the decay rate and t is the iteration index. The undesirable results may be attributed to below factors: a) originally, they only consider around 10 training tasks and randomly select tasks to train, therefore failing to deal with our more complex experimental setting; b) further designs may be needed in order to be successfully adapted to POMO, since POMO inherently improves the generalization upon AM. Moreover, the reinforcement learning setting is empirically found to be challenging for Reptile (Nichol et al., 2018). We further tune its key hyperparameters (e.g., the step size β and number of inner-loop updates K) on TSP. We follow the same experimental setups described in Section 5, and show the zero-shot performance (i.e., gaps with respect to Concorde) in Table 4. In summary, we empirically observe that a fixed and relatively large step size works better. Therefore, we report the results of Meta-POMO with $\beta = 0.9$ and $K = 50$ in Section 5.

Table 4. Tuning of Meta-POMO (Manchanda et al., 2022) on TSP.

Test Task	$K = 50$							$\beta = 0.9$			
	$\beta = 0.1$	$\beta = 0.3$	$\beta = 0.5$	$\beta = 0.7$	$\beta = 0.9$	$\beta = 0.99$	decaying β	$K = 2$	$K = 5$	$K = 10$	$K = 25$
$(300, R)$	8.53%	6.37%	5.43%	5.03%	**4.37%**	4.78%	9.03%	6.18%	6.20%	5.16%	4.89%
$(300, E)$	8.16%	5.87%	5.05%	4.76%	**4.14%**	4.50%	8.50%	5.72%	5.81%	4.78%	4.56%
$(500, R)$	14.39%	12.10%	10.91%	10.41%	**9.41%**	10.12%	15.24%	11.86%	11.91%	10.41%	10.02%
$(500, E)$	14.16%	11.69%	10.68%	10.46%	**9.44%**	10.01%	14.80%	11.55%	11.62%	10.24%	9.94%

[5] https://www.math.uwaterloo.ca/tsp/concorde/index.html
[6] http://akira.ruc.dk/~keld/research/LKH-3/
[7] https://github.com/vidalt/HGS-CVRP

D.3. Results on Benchmark Instances

We evaluate all methods on the classical benchmark datasets, such as TSPLIB[8] (Reinelt, 1991) and CVRPLIB[9] (Set-X) (Uchoa et al., 2017), where we choose representative instances with size $n \in [100, 1002]$. We also combine our method with the efficient active search (Hottung et al., 2022). Specifically, following their original implementation[10], we run EAS-Lay and EAS-Emb (with 1 run and 200 iterations) on each instance, and report the best result. Due to the huge GPU memory it needs, we only run it on instances with size $n \in [100, 750]$. The detailed results are shown in Table 5 and Table 6.

Table 5. Results on TSPLIB (Reinelt, 1991) instances.

Instance	Opt.	POMO Obj.	POMO Gap	AMDKD-POMO Obj.	AMDKD-POMO Gap	Meta-POMO Obj.	Meta-POMO Gap	Ours Obj.	Ours Gap	Ours+EAS Obj.	Ours+EAS Gap
kroA100	21282	**21282**	**0.00%**	21360	0.37%	21308	0.12%	21305	0.11%	21282	0.00%
kroA150	26524	**26823**	**1.13%**	26997	1.78%	26852	1.24%	26873	1.32%	26566	0.16%
kroA200	29368	**29745**	**1.28%**	30196	2.82%	29749	1.30%	29823	1.55%	29460	0.31%
kroB200	29437	30060	2.12%	30188	2.55%	29896	1.56%	**29814**	**1.28%**	29445	0.03%
ts225	126643	131208	3.60%	**128210**	**1.24%**	131877	4.13%	128770	1.68%	127281	0.50%
tsp225	3916	4040	3.17%	4074	4.03%	4047	3.35%	**4008**	**2.35%**	3933	0.43%
pr226	80369	**81509**	**1.42%**	82430	2.56%	81968	1.99%	81839	1.83%	81235	1.08%
pr264	49135	50513	2.80%	51656	5.13%	**50065**	**1.89%**	50649	3.08%	49212	0.16%
a280	2579	2714	5.23%	2773	7.52%	2703	4.81%	**2695**	**4.50%**	2591	0.47%
pr299	48191	50571	4.94%	51270	6.39%	49773	3.28%	**49348**	**2.40%**	48449	0.54%
lin318	42029	44011	4.72%	44154	5.06%	**43807**	**4.23%**	43828	4.28%	43090	2.52%
rd400	15281	16254	6.37%	16610	8.70%	16153	5.71%	**15948**	**4.36%**	15531	1.64%
fl417	11861	12940	9.10%	13129	10.69%	12849	8.33%	**12683**	**6.93%**	12754	7.53%
pr439	107217	115651	7.87%	117872	9.94%	114872	7.14%	**114487**	**6.78%**	111902	4.37%
pcb442	50778	55273	8.85%	56225	10.73%	55507	9.31%	**54531**	**7.39%**	53069	4.51%
d493	35002	38388	9.67%	38400	9.71%	38641	10.40%	**38169**	**9.05%**	37850	8.14%
u574	36905	41574	12.65%	41426	12.25%	41418	12.23%	**40515**	**9.78%**	39295	6.48%
rat575	6773	**7617**	**12.46%**	7707	13.79%	7620	12.51%	7658	13.07%	7333	8.27%
p654	34643	38556	11.30%	39327	13.52%	38307	10.58%	**37488**	**8.21%**	39141	12.98%
d657	48912	55133	12.72%	55143	12.74%	54715	11.86%	**54346**	**11.11%**	53077	8.52%
u724	41910	48855	16.57%	48738	16.29%	48272	15.18%	**48026**	**14.59%**	48144	14.87%
rat783	8806	10401	18.11%	10338	17.40%	**10228**	**16.15%**	10300	16.97%	–	
pr1002	259045	310855	20.00%	312299	20.56%	308281	19.01%	**305777**	**18.04%**	–	

Table 6. Results on CVRPLIB (Set-X) (Uchoa et al., 2017) instances.

Instance	Opt.	POMO Obj.	POMO Gap	AMDKD-POMO Obj.	AMDKD-POMO Gap	Meta-POMO Obj.	Meta-POMO Gap	Ours Obj.	Ours Gap	Ours+EAS Obj.	Ours+EAS Gap
X-n101-k25	27591	**28804**	**4.40%**	28947	4.91%	29647	7.45%	29442	6.71%	27750	0.58%
X-n153-k22	21220	23701	11.69%	23179	9.23%	23428	10.41%	**22810**	**7.49%**	21864	3.03%
X-n200-k36	58578	**60983**	**4.11%**	61074	4.26%	61632	5.21%	61496	4.98%	59765	2.03%
X-n251-k28	38684	**40027**	**3.47%**	40262	4.08%	40477	4.63%	40059	3.55%	39198	1.33%
X-n303-k21	21736	22724	4.55%	22861	5.18%	22661	4.26%	**22624**	**4.09%**	22035	1.38%
X-n351-k40	25896	**27410**	**5.85%**	27431	5.93%	27992	8.09%	27515	6.25%	26644	2.89%
X-n401-k29	66154	68435	3.45%	68579	3.67%	68272	3.20%	**68234**	**3.14%**	67365	1.83%
X-n459-k26	24139	26612	10.24%	26255	8.77%	25789	6.84%	**25706**	**6.49%**	25144	4.16%
X-n502-k39	69226	71435	3.19%	71390	3.13%	71209	2.86%	**70769**	**2.23%**	70277	1.52%
X-n548-k50	86700	90904	4.85%	90890	4.83%	90743	4.66%	**90592**	**4.49%**	89542	3.28%
X-n599-k92	108451	115894	6.86%	115702	6.69%	**115627**	**6.62%**	116964	7.85%	113089	4.28%
X-n655-k131	106780	110327	3.32%	111587	4.50%	110756	3.72%	**110096**	**3.11%**	108433	1.55%
X-n701-k44	81923	86933	6.12%	88166	7.62%	86605	5.72%	**86005**	**4.98%**	85432	4.28%
X-n749-k98	77269	**83294**	**7.80%**	83934	8.63%	84406	9.24%	83893	8.57%	81040	4.88%
X-n801-k40	73311	80584	9.92%	80897	10.35%	79077	7.87%	**78171**	**6.63%**	–	
X-n856-k95	88965	96398	8.35%	95809	7.69%	**95801**	**7.68%**	96739	8.74%	–	
X-n895-k37	53860	61604	14.38%	62316	15.70%	59778	10.99%	**58947**	**9.44%**	–	
X-n957-k87	85465	93221	9.08%	93995	9.98%	92647	8.40%	**92011**	**7.66%**	–	
X-n1001-k43	72355	82046	13.39%	82855	14.51%	79347	9.66%	**78955**	**9.12%**	–	

[8] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp
[9] http://vrp.galgos.inf.puc-rio.br/index.php
[10] https://github.com/ahottung/EAS

We further evaluate all methods on Set-XML100 (Queiroga et al., 2022), which is a newly proposed CVRP benchmark dataset, with the size of instances $n \in [100, 5000]$. The instances have a broader range of distribution shifts, such as depot positioning (A), customer positioning (B), demand distribution (C), and average route size (D). Since the original dataset only contains VRP100 instances, we randomly sample 5 instances and further generate 30 ($= 5 \times 6$) instances with size $n \in \{500, 1000, 2000, 3000, 4000, 5000\}$. We randomly sample the four characteristics from the Cartesian product of $A \in \{1, 2, 3\} \times B \in \{1, 2, 3\} \times C \in \{1, 2, 3, 4, 5, 6, 7\} \times D \in \{1, 2, 3, 4, 5, 6\}$. We use the given optimal solutions for CVRP100 instances, and use HGS to obtain (sub-)optimal solutions for other newly generated instances. The setting of HGS is the same as the one in Appendix D.1. The solving time varies from hours to days depends on n. The results are shown in Table 7, where each instance has the form of XML$\{n\}$_$\{ABCD\}$_{ID}. We omit the ID (i.e., 01) for simplicity.

Table 7. Results on CVRPLIB (Set-XML100) (Queiroga et al., 2022) instances.

		POMO		AMDKD-POMO		Meta-POMO		Ours	
Instance	(Sub-)Opt.	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
XML100_1113	14740	**15049**	**2.10%**	15182	3.00%	15125	2.61%	15076	2.28%
XML100_1341	24931	25927	4.00%	**25796**	**3.47%**	26560	6.53%	26143	4.86%
XML100_2271	20100	21782	8.37%	21109	5.02%	21333	6.13%	**20877**	**3.87%**
XML100_3123	20370	**20704**	**1.64%**	20978	2.98%	20907	2.64%	20883	2.52%
XML100_3372	33926	37235	9.75%	37301	9.95%	37082	9.30%	**36292**	**6.97%**
XML500_1215	37174	39302	5.72%	39152	5.32%	38817	4.42%	**38689**	**4.08%**
XML500_1246	23205	25532	10.03%	25516	9.96%	25212	8.65%	**25096**	**8.15%**
XML500_1344	47944	51257	6.91%	51452	7.32%	**50541**	**5.42%**	50657	5.66%
XML500_3134	65408	69527	6.30%	69675	6.52%	69284	5.93%	**68703**	**5.04%**
XML500_3315	44783	47556	6.19%	47595	6.28%	47294	5.61%	**47104**	**5.18%**
XML1000_1276	42095	48226	14.56%	49132	16.72%	46358	10.13%	**46342**	**10.09%**
XML1000_1335	63968	72555	13.42%	72733	13.70%	70118	9.61%	**69470**	**8.60%**
XML1000_2256	30862	36202	17.30%	36448	18.10%	34908	13.11%	**34182**	**10.76%**
XML1000_2363	85618	96685	12.93%	95985	12.11%	94893	10.83%	**93445**	**9.14%**
XML1000_3113	169377	179276	5.84%	180583	6.62%	178765	5.54%	**178171**	**5.19%**
XML2000_1172	336322	**392613**	**16.74%**	416007	23.69%	414319	23.19%	395090	17.47%
XML2000_1214	194617	209676	7.74%	211107	8.47%	206678	6.20%	**205204**	**5.44%**
XML2000_1326	69613	97656	40.28%	95535	37.24%	84193	20.94%	**83356**	**19.74%**
XML2000_2216	56550	75417	33.36%	70690	25.00%	65542	15.90%	**63906**	**13.01%**
XML2000_3316	105108	120956	15.08%	129375	23.09%	119440	13.64%	**116758**	**11.08%**
XML3000_1141	800995	**890313**	**11.15%**	980642	22.43%	938765	17.20%	910961	13.73%
XML3000_2221	615170	667875	8.57%	703465	14.35%	**656973**	**6.80%**	674764	9.69%
XML3000_2322	400934	450847	12.45%	487873	21.68%	448146	11.78%	**446922**	**11.47%**
XML3000_3155	244524	328102	34.18%	308877	26.32%	285693	16.84%	**271352**	**10.97%**
XML3000_3313	427510	471327	10.25%	488874	14.35%	467088	9.26%	**459396**	**7.46%**
XML4000_1211	1296150	1397205	7.80%	1451127	11.96%	1360158	4.94%	**1336333**	**3.10%**
XML4000_1246	149850	190303	27.00%	198247	32.30%	**173269**	**15.63%**	174495	16.45%
XML4000_2153	330364	684832	107.30%	540420	63.58%	**379186**	**14.78%**	502960	52.24%
XML4000_3161	1516100	1694469	11.76%	1805507	19.09%	1755874	15.82%	**1658308**	**9.38%**
XML4000_3246	156226	292968	87.53%	206601	32.24%	184648	18.19%	**183801**	**17.65%**
XML5000_1241	1584020	1741474	9.94%	1778777	12.30%	1826274	15.29%	**1718791**	**8.51%**
XML5000_1321	1466910	**1597897**	**8.93%**	1886757	28.62%	1677306	14.34%	1647989	12.34%
XML5000_2224	315739	386773	22.50%	403097	27.67%	**345626**	**9.47%**	352382	11.61%
XML5000_3135	396487	892760	125.17%	755735	90.61%	477590	20.46%	**449778**	**13.44%**
XML5000_3372	1135140	**1273886**	**12.22%**	1522414	34.12%	1438497	26.72%	1293027	13.91%

D.4. Ablation Study

Here, we conduct further ablation studies on hyperparameters and technical choices. For training efficiency, we run the experiments for 125K iterations (i.e., 16M instances) using the proposed first-order approximation method on TSP. The other training setups are kept the same as the ones presented in Section 5.

Ablation Study on Hyperparameters. There are several key hyperparameters in the proposed framework: 1) the number of tasks in a mini-batch B; 2) the number of inner-loop updates K; 3) the step sizes of inner-loop and outer-loop updates α, β; 4) the temperature η, which controls the entropy of the probability distribution, from which our hierarchical task scheduler samples. The results are shown in Table 8. Note that the performance comparison is based on the same number of training instances. Since the training instances in the inner-loop and outer-loop optimization are different, increasing B or K will decrease the total number of meta-model updates in the outer-loop optimization, resulting in inferior zero-shot performance. In this paper, we follow the setting in Finn et al. (2017), with $B = K = 1$. If we would like to increase B or K, 1) the step sizes α, β need to be tuned in order to achieve decent performance; 2) the advanced meta-learning algorithms (e.g., MAML with bootstrapping (Flennerhag et al., 2022)) could be used to improve training efficiency. Moreover, we could observe that carefully tuning the temperature η may further boost the performance.

Table 8. Ablation study on Hyperparameters.

B	K	α	β	η	$(300, U)$	$(500, R)$
1	1	1e-4	1e-4	1	13.51 (4.26%)	13.54 (9.29%)
1	1	1e-4	1e-3	1	13.61 (5.03%)	13.71 (10.61%)
1	3	1e-4	1e-4	1	14.04 (8.41%)	13.45 (14.61%)
3	1	1e-4	1e-4	1	14.00 (8.09%)	14.39 (16.16%)
1	1	1e-4	1e-4	0.2	**13.49 (4.13%)**	**13.53 (9.18%)**
1	1	1e-4	1e-4	5	13.62 (5.07%)	13.73 (10.80%)

Ablation Study on Optimizers. 1) *Optimizer:* when training with REINFORCE (Williams, 1992), the Adam optimizer has much better performance than SGD, as shown in Kool et al. (2018); Kwon et al. (2020). We also empirically observe the superior performance when meta-training POMO with the Adam optimizer in the out-loop optimization (where the meta-model is updated). Another question is *which optimizer should we use in the inner-loop optimization?* We conduct the ablation study pertaining to this question, and the results in Table 9 demonstrate similar performance for different optimizers in the inner loop. Therefore, in this paper, we use the same optimizer (i.e., Adam) as the one used during fine-tuning, which is more convenient than tuning different optimizers in the same framework. 2) *Meta-Gradient:* the Adam optimizer (Kingma & Ba, 2015) is known for its performance and stability, and requires fewer hyperparameters for tuning. The internal implementation of Adam incorporates bias correction, scaling and momentum. Here the question is *could it be helpful to load the information of meta-gradients to the inner-loop optimizer?* The intuition is that the information of the meta-gradient (from the outer-loop optimizer) may serve as a good initialization of the gradient for the inner-loop optimization, so that the meta-training may achieve better convergence or final performance. Based on the results in Table 9, we observe that loading meta-gradients has no improvements. Therefore we ignore it to keep the simplicity of the framework.

Table 9. Ablation study on Optimizers.

Outer-Loop	Inner-Loop	Load Meta-Gradient	$(300, U)$	$(500, R)$
Adam	Adam	×	**13.51 (4.26%)**	**13.54 (9.29%)**
SGD	SGD	×	16.56 (27.83%)	17.47 (41.16%)
Adam	SGD	×	13.51 (4.32%)	13.55 (9.37%)
Adam	Adam	✓	13.53 (4.47%)	13.60 (9.72%)

Ablation Study on Normalization Layers. Similar to Drakulic et al. (2023), we empirically observe that the choice of the normalization layer (in attention-based models (Kool et al., 2018; Kwon et al., 2020)) has a significant effect on the final performance. We denote the batch normalization without tracking the running mean and variance as *batch_no_track*. Empirically, the batch_no_track and instance normalization could achieve decent zero-shot performance, while no normalization or batch normalization (with the first-order approximation) may destabilize the meta-training. We also try the rezero normalization layer (Bachlechner et al., 2021) without observing significant improvements. Therefore, in this paper, we use batch_no_track as the default normalization layer for our method. Although the above empirical observations may only valid for attention-based models, the choice of the normalization layer may be worthy of attention for the future work.

D.5. Generalizability

To demonstrate the generalizability of the proposed framework, we further apply it to L2D[11] (Li et al., 2021). Concretely, we train a regression model (rather than a classification model) since: (1) the training of the regression model is more efficient (i.e., around 6 hrs); (2) it has better flexibility in training multiple sizes and distributions, which is quite suitable for the omni-generalization setting. We use the datasets provided by Li et al. (2021) to construct the training task set. Concretely, it contains six mixed CVRP distributions with $n \in \{500, 1000\} \times d \in \{3, 5, 7\}$, where n is the problem size and d is the cluster center. We use HGS as the subsolver, and keep the other settings the same as Li et al. (2021). During the evaluation, we set the number of runs to 1 for each instance, and set the time limit for solving each subproblem to 1s. For a fair comparison, we retrain the regression model (i.e., L2D with the batch size of 512 and 2048), and meta-train a regression model (i.e., Ours with the batch size of 512) on the training task set. We show the average cost over 10 instances on each test dataset, and the number of achieved best solutions (in brackets) among all methods. As shown in Table 10, our method could further improve the generalization of L2D when meta-training with diverse tasks in terms of sizes and distributions, demonstrating the effectiveness and generalizability of the proposed framework.

Table 10. Performance evaluation on L2D (Li et al., 2021).

	In-Distribution			*Cross-Size*			*Cross-Distribution*	*Cross-Size & Distribution*
	mixed_d3_n1000	mixed_d5_n1000	mixed_d7_n1000	mixed_d3_n2000	mixed_d5_n2000	mixed_d7_n2000	cluster_n1000	cluster_n2000
L2D (512)	142.53 (2)	119.33 (2)	**102.96 (5)**	287.64 (2)	245.55 (3)	201.84 (1)	149.43 (2)	194.70 (2)
L2D (2048)	140.90 (1)	161.61 (2)	208.95 (1)	312.07 (0)	194.84 (3)	300.44 (0)	**81.21 (4)**	265.26 (1)
Ours (512)	**80.95 (7)**	**95.53 (6)**	124.95 (4)	**101.28 (8)**	**139.44 (4)**	**96.03 (9)**	91.96 **(4)**	**97.18 (7)**

E. Discussion

Training Efficiency and Scalability. The training efficiency and scalability of the proposed framework could be analyzed from two perspectives. 1) *Meta-learning algorithm:* the second-order meta-learning algorithm is computationally expensive due to the calculation of Hessian-vector products and the needs of keeping computational graphs in memory (so that we could backward through them a second time). The improved first-order approximation method could greatly improve the training efficiency while maintaining performance. 2) *Base model:* we use POMO (Kwon et al., 2020), which is a popular autoregressive construction-based method, as our base model in Section 5. It leverages the attention mechanism and augments each training instance by starting with different nodes. Therefore, the (meta-)training of POMO is computationally expensive especially for the large-scale problems. In addition to the autoregressive construction-based methods, non-autoregressive construction-based and improvement-based methods also receive much attention in the literature, and they have the potential to mitigate the training efficiency and scalability issue. Typically, non-autoregressive construction-based methods learn an edge adjacency matrix (i.e., heat-map), from which the advanced post-hoc search strategies (e.g., Monte-Carlo Tree Search) construct solutions. Improvement-based methods could leverage decomposition (or divide-and-conquer) to first solve small-scale subproblems, and then obtain the feasible solution to the global problem. In general, they are more computationally efficient or effective to solve large-scale problems (e.g., TSP10000 in Fu et al. (2021); Qiu et al. (2022); Sun & Yang (2023) or VRP2000 in Li et al. (2021)), but at the expense of much longer inference time or extra domain knowledge (e.g., advanced search strategies). As shown in Appendix D.5, we further apply our method to L2D (Li et al., 2021), where we find its meta-training is much more efficient than that of POMO. For example, its second-order meta-training (on VRP500-1000) only takes 9 hours on a NVIDIA V100 GPU (32GB). Therefore, the choices of meta-learning algorithms and base models may have significant effects on the training efficiency and scalability of the proposed framework.

Meta-training on Pretrained Models. We would like to note that meta-training from scratch is non-trivial based on our experiments. If a pretrained model exists, it is suggested to conduct meta-training on it. Intuitively, the pretrained model could be regarded as a good initialization for the meta-learning framework (i.e., both the inner-loop and outer-loop optimization), and therefore could improve the training efficiency.

Performance on Small-Scale VRP Instances. In this paper, we mainly consider evaluating the generalization performance on large-scale instances. Here, we provide empirical results on small-scale instances, which are chosen from CVRPLIB (Set-P) (Augerat, 1995), with $n \leq 30$. The results are shown in Table 11. We observe that the performance on small instances may not be necessarily good if training on large sizes (i.e., CVRP50-200). The zero-shot performance is around

[11] https://github.com/mit-wu-lab/learning-to-delegate

Table 11. Results on CVRPLIB (Set-P) (Augerat, 1995) instances.

Instance	Opt.	Ours		Ours+EAS	
		Obj.	Gap	Obj.	Gap
P-n16-k8	450	**450**	**0.00%**	**450**	**0.00%**
P-n19-k2	212	219	3.30%	**212**	**0.00%**
P-n20-k2	216	225	4.17%	**216**	**0.00%**
P-n21-k2	211	213	0.95%	**211**	**0.00%**
P-n22-k2	216	219	1.39%	**216**	**0.00%**
P-n22-k8	603	610	1.16%	**603**	**0.00%**
P-n23-k8	529	548	3.59%	**529**	**0.00%**

0.00%-4.17% with greedy search. However, for small-scale instances, it is quite efficient to use advanced search strategies (e.g., EAS (Hottung et al., 2022)), resulting in 0.00% on all these benchmark instances.

Sensitivity on Training Task Distribution. The training task set is expected to contain instances with diverse distributions. As discussed in Appendix B, in addition to the uniform distribution, we use the gaussian mixture distribution due to its elegance and flexibility in changing distributions (via hyperparameters) into various patterns. Moreover, we also try to use different training task distributions. For example, we only use uniform, gaussian and cluster distributions, which are commonly used in the literature, to construct the training task set. Although our method outperforms baselines, all methods cannot achieve decent omni-generalization performance during evaluation. This might be led by the monotonous training tasks, which could not provide the model with sufficient or diverse information, making it hard to generalize well.