Towards Omni-generalizable Neural Methods for Vehicle Routing Problems

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE
EINDHOVEN UNIVERSITY OF TECHNOLOGY
SHANDONG UNIVERSITY 1901
SMU SINGAPORE MANAGEMENT UNIVERSITY

ICML International Conference On Machine Learning

TL;DR: We study a challenging yet realistic setting, which considers the generalization across both size and distribution (a.k.a., omni-generalization) in vehicle routing problems (VRPs). In this paper, we propose a general meta-learning framework to improve the omni-generalization of popular neural VRP solvers.

Motivation

- Learning heuristics for 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 (see Figure 1).
- While some attempts have been made to tackle the generalization issue of neural methods for VRPs, most of them solely focus on either size or distribution. 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 and CVRPLIB) may vary in both.
- Based on the *No Free Lunch Theorems of Machine Learning*, it is unrealistic to train a one-size-fits-all model that could perform well on any task. Therefore, we resort to meta-learning to learn a good initialized model for fine-tuning afterwards. We aim to improve the omni-generalization of neural VRP solvers.

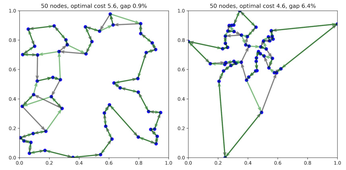

Figure 1. Generalization issue of neural solvers.

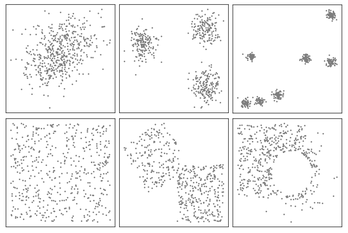

Figure 2. TSP500 instances with diverse distributions.

Methodology

Meta-Learning Framework

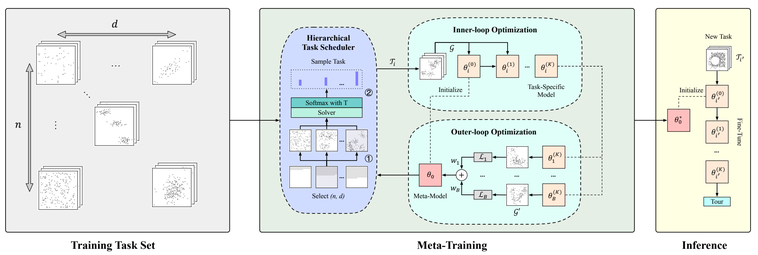

Figure 3. The illustration of the proposed framework. The training task set consists of tasks with diverse sizes $n \sim N$ and distributions $d \sim D$. In each iteration, the hierarchical task scheduler adaptively selects a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$ for meta-training, which consists 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.

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, the meta-objective is defined as follows:

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

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 $\mathcal{T}_i$. Inspired by *MAML*, to directly optimize this objective, the meta-training procedure comprises the inner-loop and outer-loop optimization at each iteration. 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.

Inner-Loop Optimization

Given a task $\mathcal{T}_i \sim p(\mathcal{T})$, we initialize the task-specific model by the meta-model θ_0, and adapt it to $\mathcal{T}_i$ by performing K gradient update steps on the training instances:

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

Outer-Loop Optimization

For each task $\mathcal{T}_i \sim p(\mathcal{T})$, the few-shot generalization performance of the task-specific model $\theta_i^{(K)}$ is evaluated on the validation instances. The meta-gradient is then obtained as follows:

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

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^*;

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

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. Instead of randomly sampling tasks for meta-training, we develop a simple yet effective hierarchical task scheduler for VRPs. Specifically, the scheduler first chooses small-size tasks and gradually transfers to large-size tasks in a curriculum way. For tasks with the same sizes, the probabilities of being selected are parameterized by their difficulties with respect to the current meta-model θ_0.

First-Order Approximation

The meta-gradient in Eq. (6) involves a gradient through a gradient (i.e., second-order derivative), and therefore it is computationally expensive to obtain due to the calculation of Hessian-vector products. Following FOMAML, the first-order approximation of the meta-model update can be expressed as:

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

However, we empirically observe that meta-training POMO from scratch with the above approximation may induce fluctuating validation performance. in order to reduce the computational cost and stabilize the meta-training, we propose to early stop the usage of second-order derivatives. Specifically, we start the meta-model updates with second-order derivatives and switch to the first-order when the optimization tends to be stable.

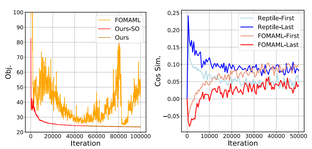

Figure 4. *Left panel:* the meta-training curves; *Right panel:* the cosine similarity between the second-order derivative and its first-order approximation at the first and last layer of the model.

Experiments

Setups:

- Baselines: We compare our method with various POMO variants. All methods are retrained with the same number of training instances.
- Zero-Shot: The solution is directly constructed using the learned policy $\pi_{\theta_0^*}$ with efficient search strategies (e.g., greedy rollout).
- Few-Shot: 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;
- Active Search: The model is adapted to each test instance by learning instance-dependent parameters.

Table 1. Evaluation on cross-size and distribution generalization.

	Method	Cross-Size and Distribution Generalization (1K ins.) (300, R) Obj. (Gap)	Time	(300, E) Obj. (Gap)	Time	(500, R) Obj. (Gap)	Time	(500, E) Obj. (Gap)	Time	(1000, R) Obj. (Gap)	Time	(1000, E) 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	28.60 (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

Table 2. Results on CVRPLIB (Set-XML100) instances.

Instance	(Sub-)Opt.	POMO Obj.	POMO Gap	AMDKD-POMO Obj.	AMDKD-POMO Gap	Meta-POMO Obj.	Meta-POMO Gap	Ours Obj.	Ours 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**	**10.75%**
XML1000_2363	85618	96805	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%

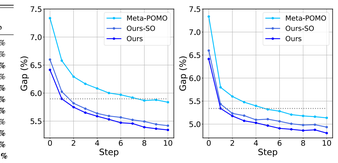

Figure 5. Experiments on efficient adaptation to CVRP (500, R) test task using (a) 100 instances; (b) 1000 instances.

Extensive experiments on both synthetic and benchmark instances demonstrate the effectiveness of our framework. We also conduct ablation studies, and generalizability exps on L2D. For more results, please check out our paper.

Welcome to check out our paper and source code for more details and information!

Paper Code