On the Generalization of Neural Combinatorial Optimization Heuristics

Sahil Manchanda[1,2⋆], Sofia Michel[1], Darko Drakulic[1], and Jean-Marc Andreoli[1]

[1] NAVER LABS Europe
[2] Indian Institute of Technology Delhi
sahil.manchanda@cse.iitd.ac.in,{sofia.michel,darko.drakulic,jean-marc.andreoli}@naverlabs.com

Abstract. Neural Combinatorial Optimization approaches have recently leveraged the expressiveness and flexibility of deep neural networks to learn efficient heuristics for hard Combinatorial Optimization (CO) problems. However, most of the current methods lack generalization: for a given CO problem, heuristics which are trained on instances with certain characteristics underperform when tested on instances with different characteristics. While some previous works have focused on varying the training instances properties, we postulate that a one-size-fit-all model is out of reach. Instead, we formalize solving a CO problem over a given instance distribution as a separate learning task and investigate meta-learning techniques to learn a model on a variety of tasks, in order to optimize its capacity to adapt to new tasks. Through extensive experiments, on two CO problems, using both synthetic and realistic instances, we show that our proposed meta-learning approach significantly improves the generalization of two state-of-the-art models.

Keywords: Neural Combinatorial Optimization · Generalization · Heuristic Learning, Traveling Salesman Problem, Capacitated Vehicle Routing Problem

1 Introduction

Combinatorial optimization (CO) aims at finding optimal decisions within finite sets of possible decisions; the sets being typically so large that exhaustive search is not an option [5]. CO problems appear in a wide range of applications such as logistics, transportation, finance, energy, manufacturing, etc. CO heuristics are efficient algorithms that can compute high-quality solutions but without optimality guarantees. Heuristics are crucial to CO, not only for applications where optimality is not required, but also for exact solvers, which generally exploit numerous heuristics to guide and accelerate their search procedure [7]. However, the design of such heuristics heavily relies on problem-specific knowledge, or at least experience with similar problems, in order to adapt generic methods to the setting at hand. This design skill that human experts acquire with experience and that is difficult to capture formally, is a typical signal for which statistical methods

⋆ Work done while interning at NAVER LABS Europe

may help. In effect, machine learning has been successfully applied to solve CO problems, as shown in the surveys [4,3]. In particular, *Neural Combinatorial Optimization* (NCO) has shown remarkable results by leveraging the full power and expressiveness of deep neural networks to model and automatically derive efficient CO heuristics.

Among the approaches to NCO, supervised learning [28,17,12] and reinforcement learning [2,20,14] are the main paradigms.

Despite the promising results of end-to-end heuristic learning, a major limitation of these approaches is their lack of generalization to out-of-training-distribution instances for a given CO problem [3,11]. For example, models are generally trained on graphs of a fixed size and perform well on unseen "similar" graphs of the same size. However, when tested on smaller or larger ones, performance tends to degrade drastically. Although size variation is the most reported case of poor generalization, in our study we will show that instances of the same size may still vary enough to cause generalization issues. This limitation might hinder the application of NCO to real-life scenarios where the precise target distribution is often not known in advance and can vary with time. A natural way to alleviate the generalization issue is to train on instances with diverse characteristics, such as various graph sizes [13,18,16]. Intuitively this amounts to augmenting the training distribution to make it more likely to correctly represent the target instances.

In this paper, we postulate that a one-size-fit-all model is out of reach.

Instead, we believe that one of the strengths of end-to-end heuristic learning is precisely its capacity to adapt to specific data and the exploitation of the underlying structure to obtain an effective specialized heuristic. Therefore we propose to use instance characteristics to define distributions and consider solving a CO problem over a given *instance distribution* as a separate *learning task*. We will assume a prior over the target task, by assuming it is part of a given *task distribution*, from which we will sample the training tasks. Note that this is a weaker assumption than most current NCO methods that (implicitly) assume knowing the target distribution at training. At the other extreme, without any assumption on the target distribution, the No Free Lunch Theorems of Machine Learning [30] tell us that we cannot expect to do better than a random policy. In this context, meta-learning [24,22] is a natural approach to obtain a model able to adapt to new unseen tasks. Given a distribution of tasks, the idea of meta-learning is to train a model using a sample of those tasks while optimizing its ability to adapt to each of them. Then at test time, when presented with an unseen task from the same distribution, the model only needs to be fine-tuned using a small amount of data from that task.

Contributions: We focus on two representative state-of-the-art NCO approaches: (i) the reinforcement learning-based method of [14] and the supervised learning approach of [12]. In terms of CO problems, we use the well-studied Traveling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP). We first analyze the NCO models' generalization capacity along different instance parameters such as the graph size, the vehicle capacity and the spatial

distribution of the nodes and highlight the significant drop in performance on out-of-distribution instances (Section 3). Then we introduce a model-agnostic meta-learning procedure for NCO, inspired by the first-order meta-learning framework of [21] and adapt it to both the reinforcement and supervised learning-based NCO approaches (Section 4).

Finally, we design an extensive set of experiments to evaluate the performance of the meta-trained models with different pairs of training and test distributions. Our contributions are summarized as follows:

- **Problem formalization:** We give the first formalization of the NCO out-of-distribution generalization problem and provide experimental evidence of its impact on two state-of-the-art NCO approaches.
- **Meta-learning framework:** We propose to apply a generic meta-training procedure to learn robust NCO heuristics, applicable to both reinforcement and supervised learning frameworks. To the best of our knowledge we are the first to propose meta-learning in this context and prove its effectiveness through extensive experiments.
- **Experimental evaluation:** We demonstrate experimentally that our proposed meta-learning approach does alleviate the generalization issue. The meta-trained models show a better zero-shot generalization performance than the commonly used multi-task training strategy. In addition, using a limited number of instances from a new distribution, the fine-tuned meta-NCO models are able to catch-up, and even frequently outperform, the reference NCO models, that were specifically trained on the target distribution. We provide results both on synthetic datasets and the well-established realistic Operations Research datasets *TSPlib* and *CVRPlib*.
- **Benchmarking datasets:** Finally, by extending commonly used datasets, we provide an extensive benchmark of labeled TSP and CVRP instances with a diverse set of distributions, that we hope will help better evaluate the generalization capability of NCO methods on these problems.

2 Related work

Several papers have noted the lack of out-of-training-distribution generalization of current NCO heuristics, e.g. [3,4]. In particular, [11] explored the role of certain architecture choices and inductive biases of NCO models in their ability to generalize to large-scale TSP problems. In [18], the authors proposed a curriculum learning approach to train the attention model of [14], assuming good-quality solutions can be accessed during training and using the corresponding optimality gap to guide the scheduling of training instances of various sizes. The proposed curriculum learning in a semi-supervised setting helped improve the original model's generalization on size. Recently, [8] proposed a method able to generalize to large-scale TSP graphs by combining the predictions of a learned model on small subgraphs and using these predictions to guide a Monte Carlo Tree Search, successfully generalizing to instances with up to 10,000 nodes. Note that both [8] and [18] are specifically designed to deal with size variation.

One can note that hybrid approaches combining learned components and classical CO algorithms tend to generalize better than end-to-end ones. For example, the learning-augmented local search heuristic of [16] was able to train on relatively small CVRP instances and generalize to instances with up to 3000 nodes. Also recent learned heuristics within branch and bound solvers show a strong generalization ability [19,31]. Other approaches that generalize well are based on algorithmic learning. For instance, [9] learns to imitate the Ford-Fulkerson algorithm for maximum bipartite matching, by neural execution of a Graph Neural Network, similar to [27] for other graph algorithms. These methods achieve a strong generalization to larger graphs but at the expense of precisely imitating the steps of existing algorithms.

In this paper we focus on the generalization of end-to-end NCO heuristics. In contrast to previous approaches, we propose a general framework, applicable to both supervised and reinforcement (unsupervised) learning-based NCO methods, and that accounts for any kind of distribution shift, including but not restricted to graph size. To the best of our knowledge, we are the first to propose meta-learning as a generic approach to improve the generalization of any NCO model.

3 Generalization properties

To analyze the generalization properties of different NCO approaches, we focus on two wide-spread CO problems: (i) the Euclidean Traveling Salesman Problem (TSP), where given a set of nodes in a Euclidean space (typically the plane), the goal is to find a tour of minimal length that visits each node exactly once; and (ii) the Capacitated Vehicle Routing Problem (CVRP), where given a depot node, a set of customer nodes with an associated demand and a vehicle capacity, the goal is to compute a set of routes of minimal total length, starting and ending at the depot, such that each customer node is visited and the sum of demands of customers in each route does not exceed the vehicle capacity. Note that the TSP can be viewed as a special case of the CVRP where the vehicle capacity is infinite.

3.1 Instance distributions as tasks

To explore the effect of variability in the training datasets, we consider a specific family $\mathcal{T}_{N,M,C,L}$ of instance distributions (tasks), indexed by the following parameters: the *graph size* N, the *number of modes* M, the *vehicle capacity* C and the *scale* L. Given these parameters, an instance is generated by the following process. When $M \neq 0$: first, M points, called the modes, are independently sampled by an ad-hoc process which tends to spread them evenly in the unit square; then N points are independently sampled from a balanced mixture of M Gaussian components centered at the M modes, sharing the same diagonal covariance matrix, meant to keep the generated points within relatively small clusters around the modes; finally, the node coordinates are rescaled by a factor L. When $M=0$: the N points are instead directly sampled uniformly in the unit square then

rescaled by L. Additionally, in the case of the CVRP problem, the depot is chosen randomly, the vehicle capacity is fixed to C and customer demands are generated as in [20]. Examples of spatial node distributions for various TSP tasks are displayed in Figure 1.

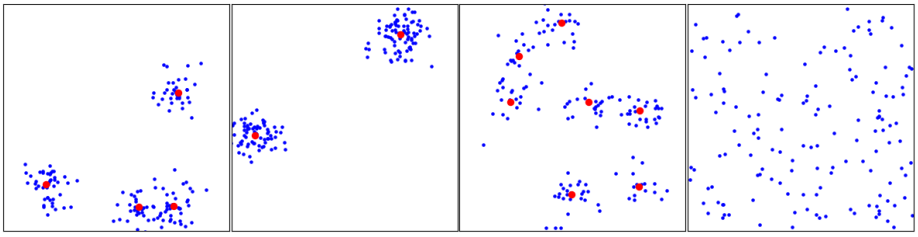

Fig. 1: A sample from each of 4 tasks $\mathcal{T}_{N=150,L=1,M}$ (blue points) with M=4, 2, 7, 0, respectively, from left to right. The red dots are the generated modes.

3.2 Measuring the impact of generalization on performance

To measure the performance of different algorithms on a given task, we sample a set of test instances from that task and apply each algorithm to each of these instances. Since the average length of the resulting tours is biased towards longer lengths, we measure instead the average "*gap*" with respect to reference tours. For the TSP, reference is provided by the Concorde solver [1], which is exact, so what we report is the true optimality gap; for the CVRP, we use the solutions computed by the state-of-the-art LKH heuristic solver [10], which returns high-quality solutions at the considered instance sizes (near optimality).

We measure the performance (gap) deterioration on generalization of the reinforcement learning based Attention Model of [14], subsequently abbreviated as AM, and the supervised Graph Convolutional Network model of [12], subsequently abbreviated as GCN. We consider several classes of tasks of the form $\mathcal{T}_{N,M,C,L}$ obtained by varying, in each class, only one of the parameters[3] N, M, C, L. For each class and each task in that class, we train each model on that task only and test it on each of the tasks in the same class, thus including the training one. The main results for the AM model are reported in Table 1.

As already observed in several papers, varying the number of nodes degrades the performance (columns (a) and (d)). Interestingly, varying the number of modes only also has a negative impact (columns (b) and (e)), and the same holds when varying the scaling of the node coordinates in the TSP (column (c)) or the vehicle capacity in the CVRP (column (f)). Similar results of performance degradation on generalization of the GCN model are given in Table 2 for TSP. These results confirm the drastic lack of generalization of both models, even on

[3] Except with CVRP where, as in previous work [20], changes to C and N are coupled.

Table 1: **Performance deterioration of AM(TSP and CVRP)**: Average gap of the AM model (in percentage, over 5000 test instances) when trained and tested on TSP instances with different (a) number of nodes N (b) number of modes M and (c) scale L; and CVRP instances with different (d) number of nodes N, (e) number of modes M and (f) vehicle capacities C.

N $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	N=20	N=50	N=100
N=20	**0.08**	1.78	22.61
N=50	0.35	**0.52**	2.95
N=100	3.78	2.33	**2.26**

(a) N (M=0, L=1)

M $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	M=0	M=3	M=6
M=0	**1.47**	32.17	2.74
M=3	26.38	**1.86**	7.32
M=6	6.91	6.01	**2.0**

(b) M (N=40, L=1)

L $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	L=1	L=5	L=10
L=1	**1.48**	282.55	292.39
L=5	32.84	**1.44**	13.83
L=10	98.62	7.12	**1.53**

(c) L (N=40, M=0)

N $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	N=20	N=50	N=100
N=20	**4.52**	12.61	20.23
N=50	7.99	**6.93**	8.47
N=100	12.90	9.75	**7.11**

(d) N (M=0, C=func(N))

M $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	M=1	M=3	M= 8
M=1	**4.39**	51.02	102.07
M=3	5.67	**6.32**	16.14
M=8	14.91	8.67	**7.85**

(e) M (N=50, C=40)

C $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	C=20	C=30	C=50
C=20	**5.83**	8.25	12.23
C=30	6.13	**7.37**	9.39
C=50	12.27	8.56	**7.99**

(f) C (N=func(C), M=0)

Table 2: **Performance deterioration of GCN(TSP)**: Average gap of the GCN model, when varying (a) the number of nodes N (b) the number of modes M and (c) the scale L.

N $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	N=20	N=50	N=100
N=20	**1.83**	38.66	77.31
N=50	22.05	**5.10**	43.76
N=100	43.86	37.26	**14.79**

(a) N (M=0, L=1)

M $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	M=0	M=3	M=8
M=0	**5.05**	35.86	26.01
M=3	35.40	**6.96**	28.71
M=8	32.74	36.29	**5.48**

(b) M (N=50, L=1)

L $\frac{\text{test}\rightarrow}{\text{train}\downarrow}$	L=1	L=5	L=10
L=1	**5.10**	28.15	32.46
L=5	272.58	**5.23**	25.41
L=10	289.51	66.28	**5.46**

(c) L (N=50, M=0)

seemingly closely related instance distributions. In the next section, we propose an approach to tackle this problem.

4 Meta-learning of NCO heuristics

The goal of this paper is to introduce an NCO approach capable of out-of-distribution generalization for a given CO problem. Since NCO methods tend to perform well on fixed instance distributions, our strategy to promote out-of-distribution generalization is to modify the way the model is trained without changing its architecture.

Concretely, given a CO problem (e.g. the TSP), we assume that we have a prior over the relevant tasks (instance distributions), possibly based on historical data. For instance, we may know that the customers in our TSP are generally clustered around city centers, but without knowing how many clusters. Our underlying assumption is that it is easier and more realistic to obtain a prior distribution on target tasks, rather than the target task itself. We propose to first train a model to learn an efficient heuristic on a sample of tasks (e.g. TSP instances with different numbers of modes). Then, considering a new unseen task

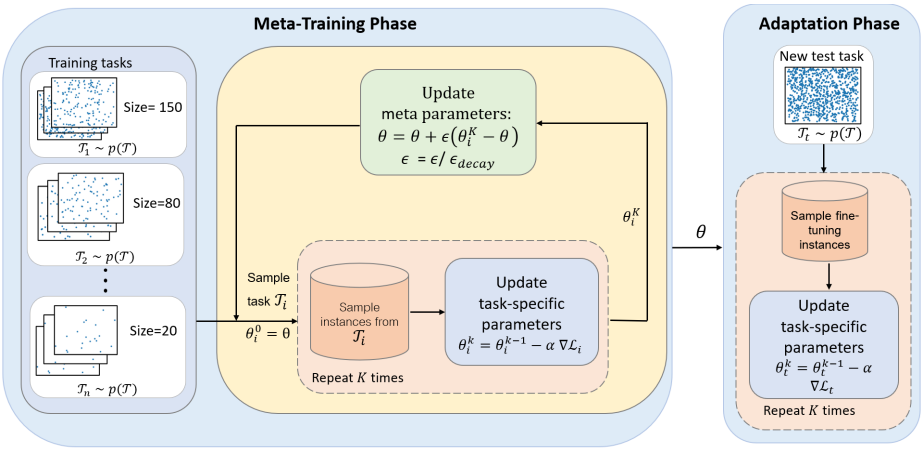

Fig. 2: Overview of our proposed method. Note that in the training phase, instead of size variation, one can have different types of distribution shifts.

(unseen number of modes), we would use a *limited number of samples* (few-shots) from that task to specialize the learned heuristic and maximize its performance on it. Fig. 2 illustrates our proposed approach.

Formally, given an NCO model with parameter θ and a distribution of tasks $\mathcal{T}$, our goal is to compute a value of θ such that, given an unseen task $t \sim \mathcal{T}$ with associated loss $\mathcal{L}_t$, after K gradient updates, the fine-tuned parameter minimizes $\mathcal{L}_t$, i.e.

$$\min_{\theta} \mathbb{E}_{t \sim \mathcal{T}}[\mathcal{L}_t(\theta_t^K)], \tag{1}$$

where θ_t^K is the fine-tuned parameter after K gradient updates of θ using batches of instances from task t. Problem (1) can be viewed as a few-shot meta-learning optimization problem. We approach it in a model-agnostic fashion by leveraging the generic Reptile meta-learning algorithm [21]. Given a task distribution, Reptile is a surprisingly simple algorithm to learn a model that performs well on unseen tasks of that distribution. Compared to the seminal MAML framework [6], Reptile is a first-order method that does not differentiate through the fine-tuning process at train time, making it feasible to work with larger values of K. And we observed experimentally that in our context, to fine-tune a model to a new task, we need up to $K = 50$ steps, which is beyond MAML's practical limits. Furthermore, since Reptile uses only first-order gradients with a very simple form, it is more efficient, both in terms of computation and memory. Using Reptile, we meta-train each model on the given task distribution to obtain an effective initialization of the parameters, which can subsequently be adapted to a new target task using a limited number of fine-tuning samples from that task.

The first step to optimize Eq. 1 consists of K updates of task specific parameters for a task $\mathcal{T}_i \sim \mathcal{T}$ as follows:

$$\begin{cases} \theta_i^0 = \theta, \\ \theta_i^k = \theta_i^{k-1} - \alpha \nabla \mathcal{L}_i(\theta_i^{k-1}), \quad \forall k \in [1 \dots K]. \end{cases} \tag{2}$$

In the above equation, the hyper-parameter α controls the learning rate. Then, using the updated parameters θ_i^K obtained at the end of the K steps, we update the meta-parameter θ as follows:

$$\theta = \theta + \epsilon \left(\theta_i^K - \theta\right). \tag{3}$$

This is essentially a weighted combination of the updated task parameters θ_i^K and previous model parameters θ. The parameter ε can be interpreted as a step-size in the direction of the Reptile "gradient" $\theta_i^K - \theta$. It controls the contribution of task specific parameters to the overall model parameters. We iterate over $\mathcal{T}_i \sim \mathcal{T}$ by computing Eq. 3 for different tasks and then using it for optimizing Eq. 1.

Scheduling ε: first specialize then generalize . As mentioned above, parameter ε controls the contribution of the task specific loss to the global meta parameters θ update in Eq. 3. A high value of ε leads to overfitting on the training task while a low value leads to underfitting, i.e. inefficient learning of the task itself. In order to tackle such scenario, in this work we utilize a simple decaying schedule for ε which starts close to 1 (i.e. we deliberately let the model forget a lot, but not all, after each new task) and tends to 0 as the training proceeds, thus stabilizing the meta-parameter that is more likely to work well for all tasks.

Fine-tuning for target adaptation: Once the model is meta-trained on a diverse set of tasks, given a new unseen task $\mathcal{T}_t$, we initialize the model parameter to the meta-trained value θ and do a number a fine-tuning steps to get the specialized parameter for that new task. Essentially,

$$\begin{cases} \theta_t^0 = \theta, \\ \theta_t^k = \theta_t^{k-1} - \alpha \nabla \mathcal{L}_t(\theta_t^{k-1}), \quad \forall k \in [1 \dots K]. \end{cases} \tag{4}$$

We can now detail the meta-training procedure of NCO models for the TSP and CVRP problems over our two state-of-the-art reinforcement learning (AM) and supervised learning (GCN) approaches to NCO heuristic learning. For simplicity, we use as default problem the TSP in this section, while the adaptation of the algorithms for the CVRP is presented in Sec. A.2 in Supplementary Material.

4.1 Meta-learning of RL-based NCO heuristics (AM model)

The RL based model of [14] (AM) consists of learning a policy that takes as input a graph representing the TSP instance and outputs the solution as a sequence of graph nodes. The policy is parameterized by a neural network with

Algorithm 1 Meta-training of the Attention Model

Require: Task set $\mathcal{T}$, # updates K, threshold β, step-size initialization $\varepsilon_0 \approx 1$ and decay $\varepsilon_{decay} > 1$

1: Initialize meta-parameters θ randomly, baseline parameters $\theta_i^b = \theta$ for $\mathcal{T}_i \in \mathcal{T}$ and step-size $\varepsilon = \varepsilon_0$
2: **while** not done **do**
3: Sample a task $\mathcal{T}_i \in \mathcal{T}$
4: Initialize adapted parameters $\theta_i \leftarrow \theta$
5: **for** K times **do**
6: Sample batch of graphs g_k from task $\mathcal{T}_i$
7: $\sigma_k \leftarrow \text{SampleRollout}(g_k, \pi_{\theta_i}) \quad \forall k$
8: $\sigma_k^b \leftarrow \text{GreedyRollout}(g_k, \pi_{\theta_i^b}) \quad \forall k$
9: $\nabla_\theta \mathcal{L}_i \leftarrow \sum_k (c(\sigma_k) - c(\sigma_k^b)) \nabla_\theta \log \pi_{\theta_i}(\sigma_k)$
10: $\theta_i \leftarrow \text{Adam}(\theta_i, \nabla_\theta \mathcal{L}_i)$ // Update for task $\mathcal{T}_i$
11: **end for**
12: **if** $\text{OneSidedPairedTTest}(\pi_{\theta_i}, \pi_{\theta_i^b}) < \beta$ **then**
13: Update baseline $\theta_i^b \leftarrow \theta_i$ // Update task specific baseline
14: **end if**
15: Update $\theta \leftarrow (1-\varepsilon)\theta + \varepsilon\theta_i, \ \varepsilon \leftarrow \varepsilon / \varepsilon_{decay}$ // Update meta parameters, step size
16: **end while**

attention based encoder and decoder [26] stages. The encoder computes nodes and graph embeddings; using these embeddings and a context vector, the decoder produces the sequence of input nodes in an auto-regressive manner. In effect, given a graph instance G with N nodes, the model produces a probability distribution $\pi_\theta(\sigma|G)$ from which one can sample to get a full solution in the form of a permutation $\sigma = (\sigma_1, \ldots, \sigma_N)$ of $\{1, \ldots, N\}$. The policy parameter θ is optimized to minimize the loss: $\mathcal{L}(\theta|G) = \mathbb{E}_{\pi_\theta(\sigma|G)}[c(\sigma)]$, where c is the cost (or length) of the tour σ. The REINFORCE [29] gradient estimator is used: $\nabla_\theta \mathcal{L}(\theta|G) = \mathbb{E}_{\pi_\theta(\sigma|G)}[(c(\sigma) - b(G)) \nabla_\theta \log \pi_\theta(\sigma|G)]$. As in [14], we use as baseline b the cost of a greedy rollout of the best model policy, that is updated periodically during training.

Meta-training of AM: Algorithm 1 describes our approach for meta-training the AM model for the TSP problem. For simplicity, the distribution of tasks that we consider here is uniform over a finite fixed set of tasks. Otherwise, one just needs to define the task-specific baseline parameters θ_i^{BL} on the fly when a task is sampled for the first time. The training consists of repeatedly sampling a task (line 3), doing K update of the meta-parameters θ using samples from that task to get fine-tuned parameters θ_i, then updating the meta-parameters as a convex combination of their previous value and the fine-tuned value (line 15). Note that the baseline need not be updated at each step (line 12), but only periodically, to improve the stability of the gradients.

4.2 Meta-learning of supervised NCO heuristics (GCN model)

The supervised model of [12] (GCN) consists of a Graph Convolution Network that takes as input a TSP instance as a graph G and outputs, for each edge ij in G, predicted probabilities $\hat{s}_{ij}$ of being part of the optimal solution. It is trained using a weighted binary cross-entropy loss between the predictions and the ground-truth solution s_{ij} provided by the exact solver Concorde [1]:

$$\mathcal{L}(\theta|G) = \sum_{ij \in G} w_0 s_{ij} \log(\hat{s}_{ij}) + w_1(1 - s_{ij}) \log(1 - \hat{s}_{ij}), \tag{5}$$

where w_0 and w_1 are class weights meant to compensate the inherent class imbalance, and B is the batch size. The predicted probabilities are then used either to greedily construct a tour, or as an input to a beam search procedure. For simplicity, and because we are interested in the learning component of the method, we only consider here the greedy version.

Meta-training of GCN: Algorithm 2 in Supplementary Material describes our approach for meta-training the GCN model. In contrast to Algorithm 1, we need here to fix the training tasks since the ground-truth optimal solutions must be precomputed in this supervised learning framework.

5 Experiments

The goal of our experiments is to demonstrate the effectiveness of meta-learning for achieving generalization in NCO. More precisely, given a prior distribution of tasks, we aim to answer the following questions: (i) How does the (fine-tuned) meta-trained NCO models perform on unseen tasks, in terms of *optimality gaps* and *sample efficiency*? (ii) How does the meta-trained models perform on unseen tasks that are *interpolated* or *extrapolated* from the training tasks? (iii) How effective is our proposed *decaying step-size* strategy in the Reptile meta-learning algorithm for our NCO tasks?

Experimental setup. Experiments were performed on a pool of machines running Intel(R) CPUs with 16 cores, 256GB RAM under CentOS Linux 7, having Nvidia Volta V100 GPUs with 32GB GPU memory. All the models were trained for 24 hours on 1 GPU. The detailed hyperparameters are presented in Sec. A.4 of the Supp. Material. Our code and datasets are available at: `https://github.com/ncometa/meta-NCO`.

Task distributions. For the TSP (resp. CVRP) experiments, we consider four task distributions (Section 3.1) which are obtained from $\mathcal{T}_{N=40,M=0,L=1}$ (resp. $\mathcal{T}_{N=50,M=0,C=40,L=1}$) as follows: (i) a *var-size* distribution is obtained by varying N only, and for training tasks within this distribution we use $N \in \{10, 20, 30, 50\}$; (ii) *var-mode* distribution by varying M only, and for training $M \in \{1, 2, 5\}$; (iii) *mixed-var* distribution by varying both N and M and training with $(N, M) \in$

$\{20, 30, 50\} \times \{1, 2, 4\}$; and (iv) only for CVRP: *var-capacity* distribution by varying C only, for training $C \in \{10, 30, 40\}$. As test tasks, we use values that are both within the training tasks range to evaluate the *interpolation* performance (e.g. M=3 for *var-mode*) and outside to evaluate the *extrapolation* performance (e.g. N=100 for *var-size*). More details about the distributions are presented in Sec. A.3 of the Supp. Mat.

Datasets. We generate synthetic TSP and CVRP instances, according to the previously described task distributions. For AM training, samples are generated on demand while for the GCN model, we generate for each task a training set of 1M instances, a validation and test set of 5K instances each and use the Concorde solver [1] and LKH [10] to get the associated ground-truth solutions for TSP and CVRP respectively (as was done in the original work). In order to fine-tune the meta-trained models, we sample a set of instances from the new task, containing either 3K (AM) or 1K (GCN) samples; these numbers were chosen as approximately 0.01% and 0.1% of the number of samples used during the 24 hours training of the AM and GCN models respectively (see details in Sec. A.5 in Supp. Mat.). In addition to synthetic datasets, we evaluate our models on the realistic datasets: TSPlib and CVRPlib. The precise settings and results are presented in Section 5.1.

Models. We use the AM-based heuristics of [14] for TSP and CVRP. For the GCN model, we use the model provided by [12] for the TSP and its adaptation by [15] for the CVRP. For a given task distribution (e.g. *var-size*) we consider the following models:

- `meta-AM` (resp. `meta-GCN`): the AM (resp. GCN) model meta-trained (following Algorithm 1 or 2 for TSP). E.g. for the *var-size* distribution, we denote this model `meta-AM-N` (resp. `meta-GCN-N`).
- `multi-AM` (resp. `multi-GCN`): the AM (resp. GCN) model trained with instances coming equiprobably from the training tasks. E.g. for the *var-mode* distribution, we denote this model `multi-AM-M` (resp. `multi-GCN-M`).
- `oracle-AM` (resp. `oracle-GCN`): original AM (resp. GCN) model trained on the *test* instance distribution, that is unseen during training of both the meta and multi models. Note that although the meta-models are not meant to improve over the oracles' performance, we will see that it happens sometimes.

To simplify the notations, we only explicitly differentiate between TSP and CVRP if it is not clear from the context. Since we are interested in the generalization of the neural models, regardless of the final decoding step (greedy, sampling, beam-search, etc), we use a simple greedy decoding for all the models. Besides, because our training is restricted to 24 hours for all the models (which is sufficient to ensure convergence of the training, see Fig.3 of Supp. Mat.), the results may not be as good are those reported in the original papers. To evaluate the impact of the meta-training on generalization when everything else fixed, we focus on the relative gap in performance between the different models.

Table 3: Average optimality gaps over 5,000 instances of the target tasks (e.g. N=100) coming from different prior task distributions (e.g. *var-size* distribution). `oracle-AM/GCN` denote the AM/GCN models trained on the target task. `multi-AM/GCN` and `meta-AM/GCN` are trained on a set of tasks from the prior distribution that does not contain the target tasks. K is the number of fine-tuning steps. In bold: for each model (AM or GCN) and each problem (TSP or CVRP), the best generalization result among the methods that were not trained on the target task.

	Tasks →	*var-size* distrib.		*var-mode* distrib.		*mixed-var* distrib.	
	Models ↓	N=100	N=150	M=3	M=8	(N,M)=(40,6)	(N,M)=(40,8)
TSP	`oracle-AM`	5.96%	12.08%	1.87 %	1.83%	2.00%	1.83%
	Farthest Ins.[23]	7.48%	**8.55%**	2.08%	2.27%	16.32%	11.70%
	`multi-AM` (K=0)	8.73%	14.40%	5.57%	6.20%	10.70%	15.18%
	`multi-AM` (K=50)	7.25%	10.87%	5.26%	4.60%	7.59%	10.26%
	`meta-AM` (K=0)	7.10%	12.25%	1.96%	2.16%	2.41%	3.50%
	`meta-AM` (K=50)	**5.58%**	9.84%	**1.82%**	**1.70%**	**2.15%**	**2.93%**
	Tasks →	*var-size* distrib.		*var-mode* distrib.		*var-capacity* distrib.	
	Models ↓	N=100	N=150	M=3	M=8	C=20	C=50
CVRP	`oracle-AM`	8.71%	11.56%	6.32 %	7.85%	5.83%	8.01%
	`multi-AM` (K=0)	18.82%	18.76%	7.87%	12.65%	9.15%	14.28%
	`multi-AM` (K=50)	9.18%	11.41%	7.58%	10.20%	8.09%	10.16%
	`meta-AM` (K=0)	11.50%	16.42%	6.05%	9.38%	6.26%	8.94%
	`meta-AM` (K=50)	**7.71%**	**9.91%**	**5.96%**	**8.45%**	**6.05%**	**8.82%**

	Tasks →	*var-size* distrib.		*var-mode* distrib.		*mixed-var* distrib.	
	Models ↓	N=80	N=100	M=3	M=8	(N,M)=(40,6)	(N,M)=(40,8)
TSP	`oracle-GCN`	12.34%	14.72%	7.65%	6.21%	6.06%	3.22%
	`multi-GCN` (K=0)	28.40%	34.29%	9.22%	7.89%	28.01%	5.05%
	`multi-GCN` (K=50)	16.73%	30.80%	8.43%	6.59%	5.99%	4.42%
	`meta-GCN` (K=0)	19.70%	32.01%	8.19%	7.32%	6.62%	3.72%
	`meta-GCN` (K=50)	**13.73%**	**18.42%**	**7.72%**	**6.45%**	**5.67%**	**3.17%**

Generalization performance: To evaluate the generalization ability of the meta-trained models, we present in Table 3 the performance of the different models at 0-shot generalization (K=0) and after K=50 fine-tuning steps, for various pairs of prior task distributions and unseen test tasks. We observe that in all cases the fine-tuned `meta-AM` clearly outperforms the fine-tuned baseline `multi-AM` and even outperforms the `oracle-AM` model in 7 out of 12 tasks.

Similar observations hold for the `meta-GCN` model: it is better both at 0-shot generalization and after fine-tuning than the `multi-GCN` baseline, and it outperforms the oracle in 2 out of 6 tasks. These results show that `meta-AM` is able to achieve impressive quality while using a negligible amount of training data of the target task compared to the original model (`oracle-AM`). More results on different target tasks as well as plots of the evolution of the performance with the number of fine-tuning steps are presented in Sec. A.7 of the Supp. Mat.

Time and sample efficiency. For a complete evaluation of the proposed meta-training and then fine-tuning approach for NCO, we discuss here its cost in terms of the fine-tuning time and number of training samples from the target task required to reach the optimality gaps of Table 3. Regarding the fine-tuning time, the 50 fine-tuning steps took 2 to 6m for `meta-AM` and 43s to 2m for `meta-GCN`. Further, generating the 1k optimal solutions for fine-tuning the supervised `meta-GCN model` took up to 17m for TSP150 and 20h for CVRP150. These values should be compared to the generation time of the 1M solutions for training the `oracle-GCN` model on the target instance distribution. Besides, for example for TSP with M=3, we observed that `oracle-AM` needs around 23 hours and more than 30 Million samples of the target task to reach the optimality gap of 1.82%. On the other hand, `meta-AM-M` only used 3000 samples from the target task and achieved a better performance after a few fine-tuning steps and less than 6 minutes. The baseline approach `multi-AM-M` was still far away at 5.2% optimality gap after fine-tuning. Similar observations hold for `meta-GCN` on TSP with M=3: `Oracle-GCN-M` needs around 22 hours and 1 Million instances of labeled data (with optimal solutions) to reach an optimality gap of 7.72%, while `meta-GCN-M` reaches the same performance in just 16 seconds, using 500 solved instances. Hence, one model trained using our prescribed meta-learning approach can be used to adapt to different tasks efficiently within a short span of time and using few fine-tuning samples. More details on training time and number of samples used for different tasks can be found in the Table 7 of the Supp. Mat. Additionally, Fig. 3 in the Supp. Mat. presents the performance of different models w.r.t time on test tasks during their course of training.

5.1 Experiments on real-world datasets

To evaluate the performance of our approach beyond synthetic datasets, we ran experiments on two well-established OR datasets: TSPlib[4] and CVRPlib[5]. From TSPlib we took the 28 instances of size 50 to 200 nodes. Note that in this context, the RL approach which does not rely on labeled data for fine-tuning is more appropriate. Since these instances are heterogeneous (i.e. no clear underlying distribution), we directly fine-tune the models on each test instance.

This is an extreme case of our setting where the target task is reduced to 1 instance. We tested the models that were (meta-)trained on the *variable-size* distribution of synthetic instances for `meta-AM` and `multi-AM`. For AM we took the pretrained model on graphs of size N=100. Because of space limitation, we grouped the instances per size range and report in Table 4 the average optimality gap obtained after K=100 fine-tuning steps, taking 20s to 1m (detailed per-instance results in Sec. A.8 of Supp Mat). Note that in this case we also fine-tune the AM model since it was not trained on the target instances distribution.

From CVRPlib we used the 106 instances of size up to 200 nodes. Since instances are grouped by sets, we apply our few-shot learning setting: fine-tuning

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

for 50 steps on approximately 10% of the instances of a set and testing on the rest. In Table 4, we report the average optimality gap over 5 random fine-tuning/test splits for each set. The results are consistent with our previous observations and illustrate the superior performance of our proposed meta-learning strategy in this realistic setting. It also shows that even if the prior task distribution is not perfect (in the sense that it does not include the target task), the meta-training gives a strong parameter initialization which one can fine-tune effectively on the target task.

Table 4: Average optimality gaps on realistic instances

Dataset→ / Model↓	**TSPlib**			**CVRPlib**				
	50−100	101−150	151−200	Set A	Set B	Set E	Set P	Set X
AM	8.52%	7.97%	17.35%	4.54%	5.69%	31.17%	5.45%	12.39%
multi-AM	11.95%	13.32%	26.04%	5.03%	5.73%	**13.00%**	6.13%	15.72%
meta-AM	**5.95%**	**5.91%**	**13.22%**	**3.56%**	**5.07%**	14.07%	**5.03%**	**11.87%**

5.2 Ablation study

Fixed vs decaying step-size ε. In this section, we study the impact of our proposed decaying ε approach during meta-training. Specifically, Table 5 presents the results of using a standard fixed step-size ε versus a decaying ε. We see that the decaying ε version of `meta-AM` and `meta-GCN` outperforms the fixed ε one, both in terms of 0-shot generalization (i.e $K = 0$) and after K=50 steps of fine-tuning. This supports our argument for performing task specialization in the beginning and generalization at the end of the meta-training procedure.

Table 5: (**Fixed vs decaying step-size ε**) Average optimality gap, on 5000 TSP instances sampled from a set of test tasks, using the meta-trained models `meta-AM` (resp. `meta-GCN`) when trained with a fixed step-size $\varepsilon = \varepsilon_0$ or a "decaying ε" where ε is close to 1 initially and tends to 0 at the end of the training.

	Test task	Fine-tuning	ε=0.1	ε=0.3	ε=0.5	ε=0.7	ε=0.9	*decaying* ε
`meta-AM`	N=100	before (K=0)	9.91%	8.33%	7.52%	6.94%	6.63%	**7.10%**
		after (K=50)	7.83%	6.50%	6.03%	5.95%	5.96%	**5.58%**
	M=8	before (K=0)	5.99%	3.07%	3.38%	2.35%	2.52%	**2.16%**
		after (K=50)	4.78%	2.27%	2.63%	1.87%	2.04%	**1.70%**
`meta-GCN`	M=6	before (K=0)	13.08%	11.90%	11.92%	12.90%	10.11%	**6.01%**
		after (K=50)	9.86%	8.27%	9.52%	10.80%	13.16%	**5.71%**
	M=8	before (K=0)	9.78%	8.81%	9.20%	11.05%	11.96%	**7.39%**
		after (K=50)	8.32%	7.37%	8.23%	9.76%	11.80%	**6.45%**

6 Conclusion

In this paper, we address the well-recognized generalization issue of end-to-end NCO methods. In contrast to previous works that aim at having one model perform well on various instance distributions, we propose to learn a model that can efficiently *adapt* to different distributions of instances. To implement this idea, we recast the problem in a meta-learning framework, and introduce a simple yet generic way to meta-train NCO models. We have shown experimentally that our proposed meta-learned RL-based and SL-based NCO heuristics are indeed robust to a variety of distribution shifts for two CO problems. Additionally, the meta-learned models also achieve superior performance on realistic datasets. We show that our approach can push the boundary of the underlying NCO models by solving instances with up to 200 nodes when the models are trained with only up to 50 nodes. While the known limitations of the underlying models (esp. the attention bottleneck, and fully-connected GCN) prevent tackling much larger problems, our approach could be applied to other models. Finally note that there are several possible levels of generalization in NCO. In this paper, we have mostly focused on improving the generalization to instance distributions for a fixed CO problem. To go further, one could investigate the generalization to other CO problems. For this more ambitious goal, domain adaptation approaches, which explicitly account for the domain shifts (e.g. using adversarial-based techniques [25]) could be an interesting direction to explore.

Acknowledgments

We wish to thank Pankaj Pansari and Anilkumar Swamy for preliminary experiments on the generalization of existing models. We are grateful to Julien Perez for helpful discussions and advice throughout the project. We also thank the anonymous reviewers for comments that helped improve the paper.

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (Sep 2011)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural Combinatorial Optimization with Reinforcement Learning. arXiv:1611.09940 [cs, stat] (Jan 2017)
3. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research **290**(2), 405–421 (2021)
4. Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. arXiv:2102.09544 [cs, math, stat] (Feb 2021)
5. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley-Interscience, New York, 1st edition edn. (Nov 1997)

6. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. pp. 1126–1135. ICML'17, JMLR.org, Sydney, NSW, Australia (Aug 2017)
7. Fischetti, M., Lodi, A.: Heuristics in Mixed Integer Programming. In: Wiley Encyclopedia of Operations Research and Management Science. American Cancer Society (2011)
8. Fu, Z.H., Qiu, K.B., Zha, H.: Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. arXiv:2012.10658 [cs] (Dec 2020)
9. Georgiev, D., Liò, P.: Neural Bipartite Matching. arXiv:2005.11304 [cs, stat] (Jun 2020)
10. Helsgaun, K.: An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems p. 60 (2017)
11. Joshi, C.K., Cappart, Q., Rousseau, L.M., Laurent, T., Bresson, X.: Learning TSP Requires Rethinking Generalization. arXiv:2006.07054 [cs, stat] (Jun 2020)
12. Joshi, C.K., Laurent, T., Bresson, X.: An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. arXiv:1906.01227 [cs, stat] (Jun 2019)
13. Joshi, C.K., Laurent, T., Bresson, X.: On Learning Paradigms for the Travelling Salesman Problem. arXiv:1910.07210 [cs, stat] (Oct 2019)
14. Kool, W., van Hoof, H., Welling, M.: Attention, Learn to Solve Routing Problems! In: International Conference on Learning Representations (2019)
15. Kool, W., van Hoof, H., Gromicho, J., Welling, M.: Deep Policy Dynamic Programming for Vehicle Routing Problems. arXiv:2102.11756 [cs, stat] (Feb 2021)
16. Li, S., Yan, Z., Wu, C.: Learning to delegate for large-scale vehicle routing. In: Advances in Neural Information Processing Systems 34 Pre-Proceedings (2021)
17. Li, Z., Chen, Q., Koltun, V.: Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 537–546. Curran Associates, Inc. (2018)
18. Lisicki, M., Afkanpour, A., Taylor, G.W.: Evaluating Curriculum Learning Strategies in Neural Combinatorial Optimization. arXiv:2011.06188 [cs] (Nov 2020)
19. Nair, V., Dvijotham, D., Dunning, I., Vinyals, O.: Learning Fast Optimizers for Contextual Stochastic Integer Programs. In: UAI 2018 (2018)
20. Nazari, M., Oroojlooy, A., Snyder, L., Takac, M.: Reinforcement Learning for Solving the Vehicle Routing Problem. In: Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
21. Nichol, A., Achiam, J., Schulman, J.: On First-Order Meta-Learning Algorithms. arXiv:1803.02999 [cs] (Oct 2018)
22. Ravi, S., Larochelle, H.: OPTIMIZATION AS A MODEL FOR FEW-SHOT LEARNING p. 11 (2017)
23. Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M.: An analysis of several heuristics for the traveling salesman problem. In: Ravi, S.S., Shukla, S.K. (eds.) Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz, pp. 45–69. Springer Netherlands, Dordrecht (2009)
24. Schmidhuber, J.: Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook (1987)
25. Tzeng, E., Hoffman, J., Saenko, K., Darrell, T.: Adversarial Discriminative Domain Adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7167–7176 (2017)

26. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention Is All You Need. arXiv:1706.03762 [cs] (Jun 2017)
27. Veličković, P., Ying, R., Padovano, M., Hadsell, R., Blundell, C.: Neural Execution of Graph Algorithms. In: International Conference on Learning Representations (Sep 2019)
28. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer Networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2692–2700. Curran Associates, Inc. (2015)
29. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3), 229–256 (May 1992)
30. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (Apr 1997)
31. Zarpellon, G., Jo, J., Lodi, A., Bengio, Y.: Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies. arXiv:2002.05120 [cs, stat] (Jun 2021)