Is Transfer Learning Helpful for Neural Combinatorial Optimization Applied to Vehicle Routing Problems?

Ali Yaddaden, Sébastien Harispe, Michel Vasquez

Is Transfer Learning helpful for Neural Combinatorial Optimization applied to Vehicle Routing Problems?

Ali Yaddaden[1 [0000−0003−0107−1188]], Sébastien Harispe[1 [0000−0001−5630−2743]], and Michel Vasquez[1 [0000−0001−6346−2550]]

EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France
{firstname.lastname}@mines-ales.fr

Abstract. Recently, combinatorial optimization problems have aroused a great deal of interest in Machine Learning, leading to interesting advances in Neural Combinatorial Optimization (NCO): the study of data-driven solvers for NP-Hard problems based on neural networks. This paper studies the benefit of Transfer Learning for NCO by evaluating how model training can be improved taking advantage of knowledge learned while solving similar tasks. We focus, in particular, on two famous routing problems: the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). The latter being a generalization of the former, we study the effect of applying Transfer Learning from a model trained to solve TSP while training a model learning to solve the Capacitated VRP (CVRP). We present adaptations of a state-of-the-art NCO model for implementing Transfer Learning. Our results based on extensive empirical experiments in different settings show that Transfer Learning may help to speed up the training process while being more sample efficient.

Keywords: Neural Combinatorial Optimization · Transfer Learning · Vehicle Routing Problem · Traveling Salesman Problem.

1 Introduction

Recent developments in deep learning and reinforcement learning have led to significant contributions in applying machine learning to create end-to-end[1] solvers for many Combinatorial Optimization problems. Numerous Neural Combinatorial Optimization (NCO) models, i.e. machine learning models based on neural networks, have thus been proposed to tackle a large variety of complex combinatorial optimization problems [13]. Interestingly, compared to traditional Operational Research approaches, such models do not rely on handcrafted and costly-to-define heuristics. They correspond to parametric models that are trained using supervised or reinforcement learning based on examples of problem instances.

[1] These approaches are to distinguish from hybrid approaches mixing machine learning models and traditional problem-specific heuristics.

Two of the ubiquitous problems widely studied under the NCO framework are the Traveling Salesman Problem (TSP) [21, 2, 10] and the Vehicle Routing Problem (VRP) [10, 7, 14]. Nevertheless, even if state-of-the-art NCO models compete with optimized solvers for TSP or VRP of small sizes, new techniques have to be proposed to make such models competing dealing with bigger instances [21, 2, 10]. In this context, one of the Machine Learning approach that deserves to be studied is Transfer Learning: an approach that can be used to take advantage of knowledge obtained solving similar tasks prior to start training a model to solve a new problem. Yet, it is known in the literature of Combinatorial Optimization that TSP and VRP are related, since the latter is a generalization of the former.[2] From this perspective, we propose to study the following question: *Does Transfer Learning from TSP to VRP under the NCO framework can be used to obtain better NCO models adapted to VRP?*

To contribute to answering this question, we evaluate Transfer Learning (TL) from TSP to VRP. This is the first study of this type to our knowledge. Our findings show that TL (i) speeds up the learning process even with relatively few instances, and (ii) improves the results (tour lengths) compared to a model trained from random weights.

The rest of the article is organized as follows: Section 2 presents the TSP and the VRP, and exposes the relationship between the two problems that make them suitable candidates for TL. The NCO framework under which comes the models that we train for our experiments as well as Transfer Learning are also introduced. Section 3 describes the model used in our experiments. Section 4 presents the experimental protocol adopted to train and test our models. Section 5 presents and discusses the main results of our experiments. Finally, Section 6 sums up our findings and gives future perspectives.

2 Literature Review

2.1 The Vehicle Routing Problem (VRP)

The Vehicle Routing Problem (VRP) is a widely studied NP-Hard combinatorial optimization problem with numerous applications in transportation and logistics. In general, the VRP considers a set of clients to which goods have to be delivered by a fleet of vehicles initially parked in depots. Each vehicle has a limited capacity of goods it can carry; this capacity can either be similar among the vehicles (homogeneous fleet) or different (heterogeneous fleet). Each client has a specific demand of goods to be delivered. The goal is then to find an optimal plan with regard to an objective function to optimize so that all clients are satisfied. The objective function to minimize can be the total distance to be traveled by the fleet of vehicles, the total traveling time, or a combination of economic costs such as the price of fuel. Constraints such as a time windows that limit the visiting

[2] Note that algorithmic advances on the TSP often lead to advances on the VRP, we can cite, for example, the $k-$opt algorithm and LKH heuristic.

time of clients to specific time intervals can also be considered. Such constraints lead to a rich family of VRP variants [19, 6].

In our study, we consider the Capacitated Vehicle Routing Problem (CVRP) which is a simple VRP variant [19]. The CVRP considers a single depot and a unique vehicle with a capacity C. In our setting, a CVRP instance (X, D) is characterized by a set of locations X and a set of demands D. X is the set of $n-1$ clients and the depot represented by their $2D$-coordinates in the Euclidean plan, i.e. $X = \{x_i \in \mathbb{R}^2\}_{i \in [\![0,n-1]\!]}$ with $i = 0$ the depot index.[3] $D = \{d_i \in \mathbb{R}\}_{i \in [\![0,n-1]\!]}$ represents the set of demands ($d_0 = 0$). X can be seen as a complete graph, with nodes being the clients and the depot, and the edges being the roads connecting the locations. All locations are connected, and the distance between two locations $(x_i, x_j)_{i,j \in [\![0,n-1]\!]}$ is defined by $\|x_i - x_j\|_2$ (L_2 norm). Finding a solution to the CVRP consists in finding T simple circuits called tours. These tours must have a minimum cost defined as the sum of the distances between the successive locations composing each tour. In addition, the following conditions must be verified (with C the capacity of the vehicle): (i) each tour begins and ends at the depot; (ii) each client belongs to only one tour ($d_i \leq C, \forall i \in [\![0, n-1]\!]$); (iii) the sum of clients' demand in a tour does not exceed C.

2.2 The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem considers a set of cities and seek for a Hamiltonian cycle with the minimum traveling distance, i.e. a visiting plan that crosses each city once and only once and returns to the departure city, such that the total travelled distance is as low as possible. It is a special case of the VRP: a TSP is a VRP with one single vehicle with an infinite capacity. In our setting, a TSP instance X consists of n connected cities represented by their $2D$-coordinates; $X = \{x_i \in \mathbb{R}^2\}_{i \in [\![0,n-1]\!]}$. It can thus be seen as a complete graph with the nodes being the cities and the edges being the roads connecting them. We here consider Euclidean TSPs, i.e. distances between cities are defined by the L_2 norm.

For the rest of this paper, we designate by *node* either a city in the TSP, or the depot or a client in the VRP.

2.3 Neural Combinatorial Optimization

Neural Combinatorial Optimization (NCO) refers to the use of deep neural networks to tackle difficult combinatorial optimization problems. Although early work exist on approaching such problems using neural networks [17], the NCO framework focuses on the recent developments based on deep learning and deep reinforcement learning. The TSP and the VRP are among the two widely studied problems within this framework, e.g. [21, 2, 5, 10, 8, 12, 15, 4]. We here provide a brief overview of the main developments in the NCO framework, assuming knowledge in Deep and Reinforcement learning. Readers can refer to the recent

[3] $[\![0, n-1]\!]$ denotes the set of integer between 0 and $n-1$ included.

survey proposed by Mazyavkina et al. for an overview of this framework applied to combinatorial optimization problems [13].

In NCO, a sequence-to-sequence model [18] is considered to map a problem instance X seen as a sequence of nodes, to the order in which each node $x_i \in X$ should appear in the tour(s); such an order is also a sequence. In an autoregressive setting, this mapping is made possible by learning a stochastic policy p_θ that computes, at each time step, a probability distribution defining the probability to select a given node. The stochastic policy is in this case a neural network parametrized by θ. Let $n \in \mathbb{N}$ be the number of nodes in an instance[4], (i.e. $|X| = n$), and $Y = (y_i)_{i \in [\![0,K]\!]}$ be a solution of a problem instance, i.e. the order in which we visit the nodes. $K = n-1$ for the TSP, $K \geq n-1$ for the VRP, since we may come back many times to the depot to refill. Under these notations, we define our policy, which gives the probability of a tour Y for an instance X using the probability chain rule as follows: $P_\theta(Y|X) = \prod_{i=0}^{K} p_\theta(y_i|y_0, ..., y_{i-1}, X)$. Each term $p_\theta(y_i|y_0, ..., y_{i-1}, X)$ is estimated using a parametric model. As we will see, different neural networks have been used to that end.

Generally, sequence-to-sequence models are made of an encoder and a decoder. The encoder is a neural network that computes a d-dimensional embedding of each node $x_i \in X$. These node embeddings are passed to another neural network called a decoder to estimate p_θ. To do so, the decoder computes a query vector that corresponds to a decoding context (i.e. a vector representation that keeps track of the nodes already selected). The query vector is used along with node embeddings in an attention block to compute the probability distribution p_θ. The attention block confronts the query vector to the node representations either with an additive module (additive attention) or a dot product module (dot product attention) to give scores. In the additive attention, the node representations and the query are processed using a linear transformation and added up together. A dot product with a parametric vector is then used to compute the attention scores. In the dot product attention, the query and the node representations are directly confronted using a dot product to give the attention scores. The given scores are then passed into a softmax to output a probability distribution. In the next paragraph, we present some of these architectures.

Pointer Network [21] is one of the seminal models used to train to solve the TSP via supervised learning. It consists of an encoder-decoder Long Short-Term Memory (LSTM) architecture with an additive attention mechanism used to learn a parametric policy to solve the TSP. This work was later on improved by training the Pointer Network with a reinforcement learning algorithm (REINFORCE) using a masking procedure that excludes already visited cities [2]. LSTM cells were later replaced by a Transformer encoder and a multi-layer perceptron decoder [5]. An encoder-decoder architecture fully based on the attention mechanism that considers a simple baseline based on a copy of the policy network has also been proposed [10]. It uses a dot product attention to compute the probability distributions. This approach can be used to solve a wide variety of routing problems, e.g. TSP and CVRP. The approach mixing Pointer Net-

[4] TSP and VRP instances have been formalized in Section 2.

work and REINFORCE [2] have been adapted to the VRP [14]. It considers a multi-layer perceptron encoder and a LSTM decoder, and introduces a masking procedure that takes into account the CVRP constraints. Other approaches consider iterative improvement methods which integrate deep neural networks [7, 3, 22].

Obtaining good quality labels for hard combinatorial optimization problems is challenging and time-consuming, this is why it is more convenient to train these approaches using reinforcement learning. Models trained that way achieve good performance on instances up to 100 nodes [10, 2]. Some approaches are even able to give good quality results on instances up to 300 nodes [7]. One main issue with reinforcement learning, especially policy gradient methods, is their sample inefficiency which require them a huge amount of interactions with the environment to perform well. In order to tackle this, we investigate whether we can save interactions for the CVRP by taking advantage of the representational capacity already obtained during learning to solve the TSP.

2.4 Transfer Learning

Transfer learning (TL) refers to transferring knowledge across different but related domains. It is a widely studied technique in supervised learning [16]; it has also proven very useful in reinforcement learning (RL) since it helps accelerate the learning process. Successful applications of TL in RL range from machine translation to Visual Question Answering and Image captioning to cite a few [23]. In TL, we consider a source domain D_s and a target domain D_t. The goal is to leverage knowledge learned from the source policy P_s to help learning the target policy P_t. There are several ways to approach TL in RL depending on many factors such as the domains differences, the transferred knowledge, the sample efficiency, etc [23, 11].

In our setting, we want to perform TL from TSP to CVRP. Since the two domains only differ in the transition dynamics (in the CVRP we can come back to the depot many times to refill while in the TSP we never visit twice any node), we will use the same policy P_s trained to solve the TSP and adapt it to the CVRP domain via RL. The following metrics will be monitored:

- Jump-start performance: indicates the initial performance of the target network. Based on that, we can deduce how sample efficient is Transfer Learning; i.e. it can either be (i) a zero-shot transfer, in that case the P_s is directly applicable to the target domain; (ii) a few-show sample that requires only few interactions from the target domain to adapt P_s or (iii) sample efficient meaning that it requires more interactions with the target domain than the few-shot setting but P_s still adapts to D_t.
- Asymptotic performance: to measure the final performance of P_t.
- Performance with fixed training epochs.

3 Model Description

This section presents technical aspects of the model and training strategy considered in our study. The approach adopted is based on the NCO architecture proposed by Kool et al. [10] which has proven to be a good candidate for both TSP and VRP. Adaptations to the original formulation have been made in order to fit the TL setting considered in this paper; they will be presented hereafter.

3.1 The Model

In our experiments, we propose to use the same number of parameters θ for solving the two problems in order to adapt them directly from TSP to VRP without the need of learning any additional parameter. To do so, we use an Attention Model [10] with few adjustments in order to fit our TL setting. This model is an encoder-decoder architecture that iteratively computes, at each step t, the probability of choosing a node over the others as described in Section 2.3.

In the original implementation, the inputs of the model for the TSP are the cities $2D$-coordinates; for the VRP in addition to the clients $2D$-coordinates there is their respective demand. In our formulation, in order to avoid introducing additional parameters to the model, we use for both problems only the $2D$-coordinates. Demands are only used in the masking operation, as it will be described later. In the seminal work of Nazari et al. [14], the authors mention that using the demands along with the embedding of the $2D$-coordinates for the decoder brings no significant improvement, thus we pushed further this idea by only using the clients' coordinates in the whole encoder-decoder approach.

Figure 1 shows the overview of the Attention Model architecture with a linear embedding followed by an encoder and a decoder that we detail below.

3.2 The Encoder

The encoder is made of two main components: an embedding layer which maps the $2D$-coordinates to a vector in a $d-$dimensional feature space, and a transformer encoder [20] that acts like a graph neural network on instances (recall that here we consider an instance as a complete graph).

The embedding layer consists of a linear transformation of the node's $2D$-coordinates, following the equation: $h_i = W_x \cdot x_i + b_x \quad \forall i \in [\![0, n-1]\!]$, with $h_i, b_x \in \mathbb{R}^d$, $x_i \in \mathbb{R}^2$ (column vector), and $W_x \in \mathbb{R}^{d \times 2}$. Unlike the original model, we do not use a separate linear embedding for the depot. Although it is a special node in the VRP, using the same linear embedding used for the clients aims at benefiting from the TL's representation of the coordinates learned while solving the TSP.

For the transformer encoder, it consists of N layers of similar transformer blocks. The first block takes as input the resulting embeddings of the $2D$-coordinates, while the others take the output of the previous layer. Each transformer block consists of two components: a multi-head attention and a feed-forward layer. We used 3 transformer blocks with 8 heads, as suggested in the

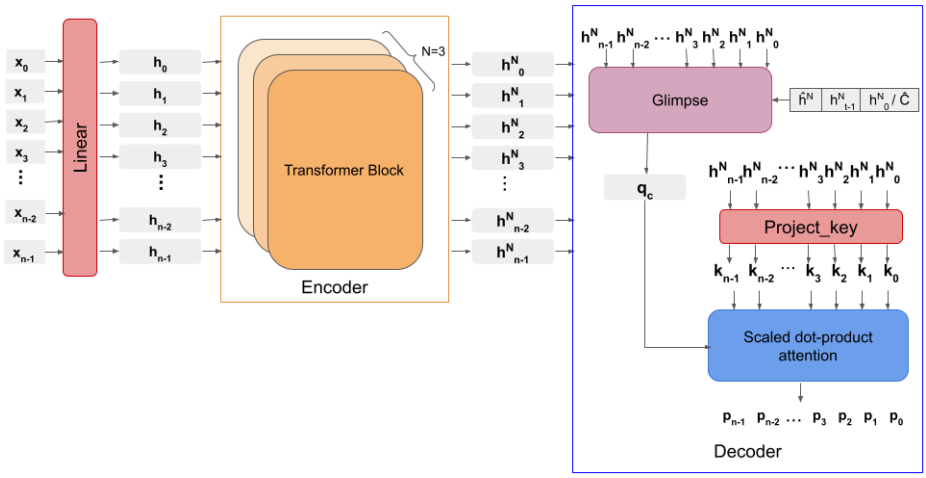

Fig. 1. Overview of the Attention Model architecture with its main components.

literature [10]. The result of the encoder is a set of $d-$dimensional node embeddings; the final embedding of a node x_i is denoted h_i^N.

3.3 The Decoder

The decoder is used to compute the probability distribution that will provide the probability that a partial solution has to be extended by a given node. As we will explain hereafter, this probability distribution will be computed confronting a context embedding with the node embeddings. In the original model, the number of parameters defining the context embedding varies whether we are dealing with TSP or VRP. Here, we also introduce modifications in order to preserve the same number of parameters for both problems.

For the TSP, at each time step t, the context embedding h_c is defined as a $(3 \times d)-$dimensional vector resulting from the concatenation of: (i) the representation of the graph ($\overline{h}^N = \frac{1}{n}\sum_{i=0}^{n-1} h_i^N$), (ii) the representation of the lastly selected node $h_{y_{t-1}}$, and (iii) the representation of the first selected node h_{y_0}. At $t = 0$, two parameter vectors $v^l \in \mathbb{R}^d$ and $v^f \in \mathbb{R}^d$ are used to substitute the undefined values of (ii) and (iii). The context embedding is thus defined by $h_c = \left[\overline{h}^N, h_{y_{t-1}}, h_{y_0}\right]$ when $t \geq 1$, and $h_c = \left[\overline{h}^N, v^l, v^f\right]$ when $t = 0$.[5]

For the VRP, the context embedding h_c is the concatenation of the representation of the graph $\overline{h}^N$, the representation of the lastly selected client (at $t = 0$, we use the representation of the depot h_0^N) and the representation of

[5] $[.,.,.]$ denotes the concatenation operation.

the remaining capacity. In order to keep the same $(3 \times d)-$dimensional embedding for the VRP, we did not use the remaining vehicle capacity C using the strategy considered in [10] - which results in a $(2 \times d + 1)-$dimensional vector. Instead, we compute an embedding of the remaining capacity $\hat{C} = W_c \cdot C$, with $C \in \mathbb{R}, W_c \in \mathbb{R}^{d \times 1}$: $h_c = \left[\overline{h}^N, h_{y_{t-1}}, \hat{C}\right] \quad \forall t$.

A glimpse mechanism [2] is then used to enhance the context vector into a $d-$dimensional query vector $q_{(c)}$ that will be used by an attention mechanism to compute the probability of selecting the next node. This mechanism consists of a multi-head attention that computes the context vector interaction with regard to the nodes that have not yet been selected. Finally, a scaled dot product attention mechanism [20] is used to compute the probability of choosing a node over another. This dot product uses the query vector $q_{(c)}$ and the keys $k_i = W^K \cdot h_i^N \quad \forall i \in [\![0, n-1]\!]$ that result from a linear projection of the node embeddings.

A mask is used to prevent selecting already selected nodes. The mask mechanism differs for TSP and VRP. For the TSP, it is straightforward as we mask already visited cities. For the VRP, the masking policy is more complex. Computing the probability distribution defining the node to visit at t, we mask: (i) clients with demands greater than the remaining capacity; (ii) already satisfied clients; (iii) the depot if it has been selected at $t-1$, to prevent selecting it at t.

3.4 Model Training

We trained the model using Reinforcement learning (RL) as it tends to express interesting generalization properties [9] that may benefit Transfer Learning.[6] The parametric policy is more particularly trained using REINFORCE with a greedy rollout baseline as used in Kool et al. [10]. Given nodes' coordinates of a TSP or VRP instance X, the loss function $\mathcal{L}$ used during training is the expectation of tour lengths considering a tour generation strategy ϕ taking advantage of the parametric policy P_θ: $\mathcal{L}(\theta|X) = \mathbb{E}_{Y \sim \phi(P_\theta(\cdot|X))}[L(Y|X)]$.

During training, the tour generation strategy ϕ is set to be the sampling strategy, i.e. at each step the selected city is sampled according to the current estimation provided by the stochastic policy P_θ. Minimization of $\mathcal{L}$ is performed using a gradient-based approach, namely REINFORCE with baseline st.:

$$\nabla \mathcal{L}(\theta|X) = \mathbb{E}_{Y \sim P_\theta(\cdot|X)} \left[(L(Y|X) - b(X)) \nabla log\ P_\theta(Y|X) \right]$$

$b(X)$ is a parametric baseline used to predict tour lengths. These predictions enable to reduce the variance of the gradient estimation. The baseline is also a deep neural network with the same architecture used for defining p_θ. Using b, tour lengths are however computed using a greedy strategy: at each step the node maximizing the probability is selected. During training, baseline parameters are obtained copying policy parameters (i.e. θ). Updates of baseline parameters are

[6] Supervised learning, e.g. with optimal labels obtained *via* the Concorde solver [1] can also be used for TSP. RL however does not require labelled data that may be time-consuming to obtain.

performed after each epoch if the improvement of the current policy over the baseline is significant (according to a paired t-test comparing current baseline and current training policy, both using a greedy strategy). Details of the training approach adopted in our experiment are provided in the paper introducing the original approach adapted in our work to fit our TL setting [10].

4 Experimental Protocol

This section presents the experimental protocol defined in our study.

4.1 Model Pre-training on TSP

We first pre-trained our model to solve the TSP using the approach defined in Section 3.4. We used the same data generation and training protocol commonly used in the literature, e.g [10]; for each training epoch, 2500 batches of 512 instances have been generated (1.28M instances per epoch). The data generation process consists of uniformly drawing $2D$-coordinates from the $[0, 1]^2$ unit square [2]. For the validation, we used 10k instances sampled from the same uniform distribution. Two models named TSP20 and TSP50 have been trained considering different datasets composed of instances of 20 and 50 nodes respectively. Both models have been trained during 100 epochs using the Adam optimizer with a learning rate of 10^{-4}. These models reach state-of-the-art performance reported in Kool et al. [10]. We also save the optimizer's state so that we can later load it for adapting the model's parameters for the VRP performing TL.

4.2 Transfer Learning to VRP

Settings commonly considered evaluating TL have been adopted in our experiments. Indeed, TL is preferred when the destination task has less training data than the source task, but the tasks are similar enough so that our model captures a representation that can be useful to help tackle the problem. We therefore used less training data than the usual training protocol. Three different settings have been tested: 16k, 32k and 64k instances per epoch, and models have only been trained for 50 epochs, thus resulting on models trained on 800k, 1.6M and 3.2M instances respectively. Considering this setting, our models are trained on respectively 160, 80 and 40 times fewer data than the original models.

Comparisons have been made considering different models trained with and without TL using the same amount of data and number of epochs. When no TL is applied, the models are trained from randomly initialized parameters using Adam optimizer with a learning rate of 10^{-4} (see [10]). When TL is applied, the learning rate is set to 10^{-5} in order to exploit the optimizer's saved state from the TSP, and to gradually adapt the model's parameters to the VRP.

VRP instances have been generated using standard protocols [14, 10], sampling $2D-$coordinates uniformly from the unit square $[0, 1]^2$. The demands d_i of

clients ($i \in [\![1, n-1]\!]$) are uniformly drawn from $[1, 9] \subset \mathbb{N}$. Vehicle capacities are 30 for instances with 20 nodes and 40 for instances with 50 nodes.

We conducted 3 series of experiments:

1. **Same**: Training a VRP model with a pre-trained TSP model trained on instances with the same size and data generation distribution.
2. **Diff size**: Training a VRP model with a pre-trained TSP model trained on instances with different size and the same data generation distribution.
3. **Diff dist**: Training a VRP model with a pre-trained TSP model trained on instances with the same size and a different data generation distribution.

5 Results and Discussion

This section presents and discusses the results obtained in our experiments.[7]

5.1 Pre-training Results

Two models, respectively denoted TSP20 and TSP50, have been trained on instances of 20 and 50 cities during pre-training on TSP. The learning curves that have been obtained are similar to those obtained for the original implementation, and average tour length performances obtained by our model are on par with the original model [10].[8]

5.2 Transfer Learning Results

We now present the results obtained in the different settings.

Same - Applying TL using pre-trained models trained on TSP with the same size and data generation distribution as the VRP instances. Figure 2 shows the average tour length of VRP models in validation with greedy decoding. Both models are presented: (i) trained from scratch with a random weight initialization (NO-TL in blue), and (ii) trained starting from a pre-trained TSP model (TL in orange). Plots a-b-c correspond to VRP20 with TL from TSP20 models, while plots d-e-f correspond to VRP50 models with TL from TSP50 models. Each plot corresponds to a number of instances per epoch used to train our model, respectively from left to right, 16k, 32k and 64k.

Three phases can globally be distinguished in the learning process:

[7] See https://github.com/AYaddaden/TL_TSP_VRP for source code and additional documentation.

[8] Tested on 10k TSP instances with 20 and 50 cities and report the average tour lengths obtained using greedy decoding. The original model achieves tour lengths of 3.85 and 5.80 for TSP 20 and TSP50 [10]. Our model respectively achieves 3.84 and 5.82.

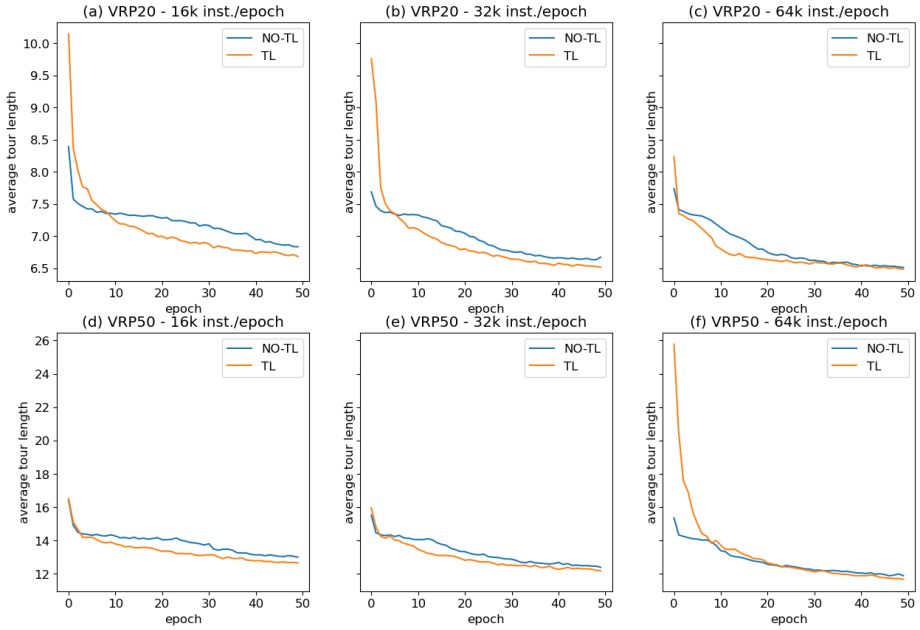

Fig. 2. Comparison of the evolution average tour lengths per epoch in validation between VRP models trained without Transfer Learning (NO-TL/blue) and VRP models trained using TL (TL/orange) with different number of instances per epoch (16k, 32k, 64k) with 20 nodes (plots a, b and c) and 50 nodes (plots d, e and f).

- Epoch 0 to 5: Initial phase, the learning curve is rather in favor of models without TL, with a rapid decrease of tour lengths for both types of models.
- Epoch 5 to 30: There is a shift in the training curves in favor of the TL models. The average tour lengths per epoch decreases faster using TL. The shift happens early in the learning process, so it seems that our models need few-shot samples to adapt to the VRP domain. Results however differ for VRP50 with 64k instances per epoch.
- Epoch 30 to 50: the learning curves of the two models continue to decrease, but more slowly than in the previous phase. TL models still outperform models without TL.

Based on our findings, it appears that for the jump-start performance metric, our models do not perform zero-shot adaptation since the initial performance is poor compared to the NO-TL models. This can be partially explained by the fact that the context embedding h_c differs for the TSP and the CVRP. The early shift seems to indicate that our models can adapt in a few-shot way, especially when there are fewer instances per epoch. For 64k instances/epoch models, our results suggest that they are still sample efficient, but with no significant improvement.

The curves also show that the more data we have, the better and faster is the convergence of the models. The asymptotic performance between the two models

gets close to each other by augmenting the number of instances per epoch. But the gap between the TL and NO-TL models is more significant when we have less training data, which means that in this setting, we can achieve better average tour lengths by considering a pre-trained model and Transfer Learning.

For our last metric, training on 50 epochs showed that for 16k and 32k instances/epoch, TL models outperform NO-TL models while for the 64k instances/epoch models, it is still difficult to decide between doing Transfer Learning and not. At least, it slightly improves the learning process.

Diff size — using a pre-trained TSP model trained on instances of sizes that are different from the VRP instances used to train our VRP model. We took as a comparison baseline our models trained with 32k instances per epoch as they are a good trade off between training time and average tour lengths achieved after 50 epochs. We tested two settings: (i) when the pre-trained TSP model was trained on instances with size smaller than the VRP's instances and (ii) when the pre-trained TSP model was trained on instances with size bigger than the VRP instances. We report results on two experiments: a VRP20 model trained on VRP instances with 20 nodes using a TSP50 pre-trained model and a VRP50 model trained on VRP instances with 50 nodes using a TSP20 pre-trained model.

Figures 3-a, 3-b show the evolution of average tour length in validation for VRP20 and VRP50, respectively. Using a pre-trained TSP model trained on instances with the same size as the VRP instances brings better average tour lengths. In Figure 3-a, for the VRP20 model, we can see that using a pre-trained TSP50 model (TL-TSP50) gives a jump-start performance similar to the model with no Transfer Learning (NO-TL) and better than the model that uses the TSP20 pre-trained model (TL-TSP20). In phase 2 (from epoch 5 to 30), we can see that Transfer Learning brings an improvement in average tour lengths in both settings (curves orange and greed are both under the blue curve). However, we can notice that the average tour lengths are better when we use a TSP20 pre-trained model. In phase 3 (epoch 30 to 50), the model with a TSP20 pre-trained model is still better and leads to a faster learning and brings better asymptotic performance. Surprisingly, at this phase, the TL-TSP50 model behaves exactly as the NO-TL model with no significant difference in asymptotic performance between the two models. In the VRP50 case (Figure 3-b), we observe that using a TSP20 pre-trained model gives worse jump-start performance than using a TSP50 model or not using Transfer Learning at all. The asymptotic performance are on par with the NO-TL model while being slightly in favor of the latter.

Our findings suggest that using a pre-trained model trained on TSP instances bigger than the VRP instances is more efficient than using a pre-trained model trained on smaller instances than the VRP instances, since there is a region in phase 2 where it behaves better than not using pre-training. Especially if we have a limited time budget to train (number of epochs), it is still beneficial to use Transfer Learning and in any case, it does not harm the learning process as similar asymptotic performances are obtained in the worst case. Intuitively, we would expect that using a TSP50 model to pre-train for VRP20 would bring a better improvement to the learning process or that using a TSP20 for pre-

training for VRP50 would be better than starting learning from scratch, but it appears based on our results, that it is not the case. We can suppose that while learning to solve the TSP on a specific instance size, the size of these instances is somehow encoded in the learned policy and that it influences its behavior.

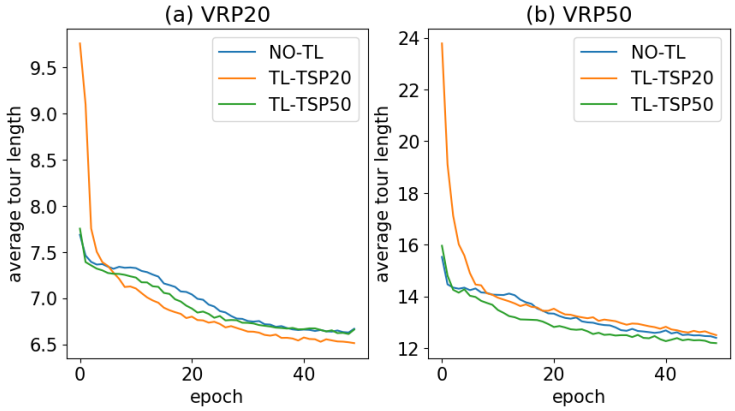

Fig. 3. Average tour lengths for VRP models, (a)- VRP20 and (b)-VRP50, trained without Transfer Learning (NO-TL/blue), using a TL with pre-trained TSP20 model (TL-TSP20/orange) and TL with pre-trained TSP50 model (TL-TSP50/green).

Diff dist — training a VRP model using instances generated from a different distribution than the distribution used to generate the TSP instances for the pre-trained models. Note that our TSP pre-trained models were trained on instances generated from a uniform distribution. We generate clients' coordinates of the VRP instances using a truncated normal distributionso that all city coordinates are inside the unit square. The demands are generated the same way as in the uniform case. Figures 4-a and 4-b show a comparison between average tour lengths of models trained from scratch (NO-TL) and with a pre-trained model (TL) with respectively 20 and 50 nodes.

Surprisingly, as we can see in Figure 4-a, using a pre-trained TSP20 model to train a VRP20 model brings an improvement in terms of speed of learning even if they are trained on instances coming from different distributions. The asymptotic performance of the TL model is better than the one of the NO-TL model. The model is able to achieve the performance of NO-TL in approximately half the epochs. From the jump-start performance perspective, we can see that neither the VRP20 nor the VRP50 can do zero-shot transfer. While the VRP20 is able to adapt in a few-shot way, the VRP50 struggles to do so. Figure 4-b shows that the pre-training does not bring significant improvement for learning a representation that would help tackle the VRP50, at least for a 50 epochs training. Indeed, while the asymptotic performance are on par with the NO-TL model, we can suppose that a shift can happen after the 50^{th} epoch in favor of the TL model. We can hypothesize that the size of the problem to tackle makes the learning process more difficult. There is probably an inductive bias induced in the learning process of TSP50 from the instances' distribution (how the cities

are spread in the unit square) that is not helpful for solving the VRP50. This bias may be less significant when the size of the instances is small.

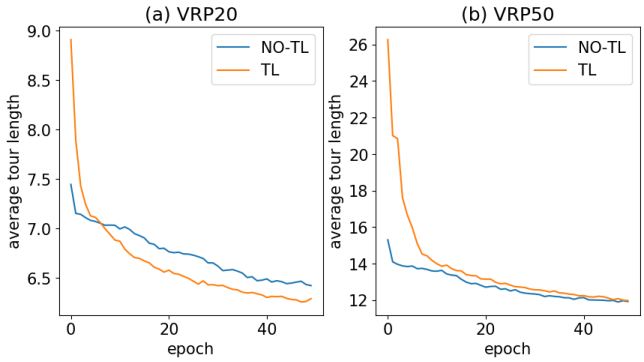

Fig. 4. Average tour lengths per epoch in validation between VRP models trained without Transfer Learning (NO-TL/blue) and VRP models trained using TL (TL/orange) with 20 nodes, (a) VRP20 and 50 nodes (b) VRP50.

6 Conclusion and Perspectives

In this paper, we have presented an empirical evaluation of Transfer Learning (TL) from TSP to CVRP in the context of Neural Combinatorial Optimization (NCO). Three metrics - jump-start, asymptotic and fixed training epochs performance - have been studied under three different training settings varying according to the instances distribution and the instance sizes of the pre-trained models. We observe that, applying NCO to CVRP, TL from TSP (i) speeds up the learning process even with relatively few instances, and (ii) improves the results (tour lengths) compared to a model trained from random weights. Our results also stress that in the worse settings, TL does not harm the learning process as asymptotic performances similar to those obtained by models with no pre-training are achieved. We also note that results vary depending on the size of the instances used in the pre-training phase: better models are obtained using pre-trained models trained on TSP instances with the same size.

Future work is required to investigate how to create models that are agnostic to the size of instances. Also, training on VRP instances sampled from a different distribution than the TSP instances proved to be more challenging when using a pre-trained model; it does work for small instances but struggles to scale to bigger instances. We encourage future contributions to investigate this specific aspect of TL for NCO applied to VRP. Indeed, if we can pre-train and train models on TSP and VRP instances sampled from different distributions, this would lead to significant improvements since it would be possible to pre-train on synthetic instances to next efficiently train the model on real-world VRP instances. Finally, it would also be interesting to investigate the application of the TL from TSP and/or CVRP to other VRP variants.

Acknowledgements. This work was granted access to the HPC resources of IDRIS under the allocation 2021-AD011011309R1 made by GENCI.

References

1. Applegate, D., Bixby, R., Chvatal, V., Cook, W.: Concorde tsp solver (2006)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940 (2016)
3. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. Advances in Neural Information Processing Systems **32**, 6281–6292 (2019)
4. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. arXiv:1704.01665 (2017)
5. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M.: Learning heuristics for the tsp by policy gradient. In: International conference on the integration of constraint programming, artificial intelligence, and operations research. pp. 170–181. Springer (2018)
6. Golden, B.L., Raghavan, S., Wasil, E.A.: The vehicle routing problem: latest advances and new challenges, vol. 43. Springer Science & Business Media (2008)
7. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. arXiv:1911.09539 (2019)
8. Joshi, C.K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. arXiv:1906.01227 (2019)
9. Joshi, C.K., Laurent, T., Bresson, X.: On learning paradigms for the travelling salesman problem. arXiv:1910.07210 (2019)
10. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv:1803.08475 (2018)
11. Lazaric, A.: Transfer in reinforcement learning: a framework and a survey. In: Reinforcement Learning, pp. 143–173. Springer (2012)
12. Ma, Q., Ge, S., He, D., Thaker, D., Drori, I.: Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. arXiv:1911.04936 (2019)
13. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. arXiv:2003.03600 (2020)
14. Nazari, M., Oroojlooy, A., Snyder, L.V., Takáč, M.: Reinforcement learning for solving the vehicle routing problem. arXiv:1802.04240 (2018)
15. d O Costa, P.R., Rhuggenaath, J., Zhang, Y., 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)
16. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering **22**(10), 1345–1359 (2009)
17. Smith, K.A.: Neural networks for combinatorial optimization: a review of more than a decade of research. INFORMS Journal on Computing **11**(1), 15–34 (1999)
18. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. arXiv:1409.3215 (2014)
19. Toth, P., Vigo, D.: The vehicle routing problem. SIAM (2002)
20. 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 (2017)
21. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. arXiv:1506.03134 (2015)
22. Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A.: Learning improvement heuristics for solving routing problems. arXiv:1912.05784 (2019)
23. Zhu, Z., Lin, K., Zhou, J.: Transfer learning in deep reinforcement learning: A survey. arXiv:2009.07888 (2020)