# Neural TSP Solver with Progressive Distillation

**Dongxiang Zhang[1], Ziyang Xiao[1], Yuan Wang[2*], Mingli Song[1], Gang Chen[1]**

[1] College of Computer Science and Technology, Zhejiang University
[2] School of Business, Singapore University of Social Sciences
{zhangdongxiang,22021206,brooksong,cg}@zju.edu.cn, wangyuan@suss.edu.sg

## Abstract

Travelling salesman problem (TSP) is NP-Hard with exponential search space. Recently, the adoption of encoder-decoder models as neural TSP solvers has emerged as an attractive topic because they can instantly obtain near-optimal results for small-scale instances. Nevertheless, their training efficiency and solution quality degrade dramatically when dealing with large-scale problems. To address the issue, we propose a novel progressive distillation framework, by adopting curriculum learning to train TSP samples in increasing order of their problem size and progressively distilling high-level knowledge from small models to large models via a distillation loss. In other words, the trained small models are used as the teacher network to guide action selection when training large models. To accelerate training speed, we also propose a Delaunary-graph based action mask and a new attention-based decoder to reduce decoding cost. Experimental results show that our approach establishes clear advantages over existing encoder-decoder models in terms of training effectiveness and solution quality. In addition, we validate its usefulness as an initial solution generator for the state-of-the-art TSP solvers, whose probability of obtaining the optimal solution can be further improved in such a hybrid manner.

## Introduction

Travelling salesman problem (TSP) determines the shortest circuit passing through $n$ given cities whose pair distance is known in advance, and requires that each city is visited exactly once. As a classical combinatorial optimization problem, TSP has been intensively studied for decades. When the problem size is small, TSP can be easily solved to derive exact solutions (Laporte 1992). In large-scale problems, exact approaches fail to tackle the exponential search space and approximate methods are adopted so as to achieve a satisfactory solution with acceptable running time. As reviewed in (Laporte 1992; Bektas 2006), there have emerged an enormous number of heuristic algorithms that trade optimality for efficiency, such as genetic algorithm, simulated annealing, tabu search, adaptive large-neighborhood search and ant colony optimization. They rely on devising effective search space exploration strategies and achieve promising results with affordable computation cost.

Recently, the adoption of encoder-decoder models to solve TSP has shed new light into this classic topic of combinatorial optimization. The design principle is that the encoder is used for effective feature extraction from input, and a solution in the form of a sequence of cities is generated iteratively by the decoder. In this way, the expensive search space exploration is avoided and the complexity of a TSP solver becomes polynomial to the problem size. A solution can then be returned instantly for real-time decision making. Such desirable property has attracted researchers' attention and various models have been proposed (Vinyals, Fortunato, and Jaitly 2015; Bello et al. 2017; Nazari et al. 2018; Deudon et al. 2018; Kool, van Hoof, and Welling 2019; Ma et al. 2019; Peng, Choi, and Xu 2021; Luo et al. 2022). Despite the encouraging achievement, the current encoder-decoder models still fail to conquer large-scale instances with hundreds of cities, in which the training process exhibits patterns of high complexity and slow convergence. Furthermore, the reward estimated from the final cost of sampled solutions is heavily delayed and may not be a reliable signal to guide action selection. In our offline experiments for TSP100, it roughly requires more than 20 million training samples and 50 hours for these models to start getting converged.

To alleviate the issue of high training barrier, we propose a novel progressive distillation framework which adopts curriculum learning to train TSP samples in increasing order of TSP problem size. In the meanwhile, the trained small models are used as teacher network to guide action selection when training large models. In this way, we can progressively distill high-level knowledge from small models to large models, via a distillation loss to minimize the difference between the teacher and student networks. Finally, to accelerate training speed, we propose a Delaunary-graph based action mask to exclude cities far away from the current decision context, and a new attention-based decoder to reduce decoding cost. Experiments results establish clear advantages of our approach over existing encoder-decoder models. We also examine its usefulness as an initial solution generator for state-of-the-art TSP solvers. The results show that with a better starting point, their probability of obtaining the optimal solution can be further

*Corresponding author

improved. The datasets and source code are provided in the supplementary materials.

## Related Work

This section focuses on the review of neural TSP solvers and we categorize them into *learning-to-generate* and *learning-to-improve* methods.

### Learning-to-Generate Methods

The learning-to-generate methods treat TSP as a sequence generation problem. Pointer network (PtrNet) (Vinyals, Fortunato, and Jaitly 2015) is an early work that devises an attention-based decoder to iteratively select an item from the input and directly outputs the city visit order. In (Joshi, Laurent, and Bresson 2019), an MLP classifier is built to compute the probability of each city pair occurring in the optimal tour, and then beam search is applied for tour inference. The drawback of these two supervised models is that their training heavily relies on high-quality labels, which are expensive to harvest for large-scale TSP instances.

To reduce the reliance on training labels, reinforcemnt learning (RL) is adopted in subsequent works. In (Bello et al. 2017), the original supervised training in PtrNet is replaced by RL and the length of the derived tour is used as a reward signal for parameter update. In (Nazari et al. 2018), the authors discard the RNN-based encoder used in PtrNet and directly uses raw embedding, i.e., $d$-dimensional city vectors generated by linear embedding. In (Deudon et al. 2018), the authors also concur the needlessness of adopting RNN in the encoder and resort to the attention mechanism proposed in Transformer (Vaswani et al. 2017). (Kool, van Hoof, and Welling 2019) shares the same idea with (Deudon et al. 2018) to rely on pure attention for input encoding and they both adopt Transformer (Vaswani et al. 2017) as the core attention mechanism. However, it does not apply positional encoding in the original Transformer because the resulting node embeddings are invariant to the input order. The output decoder is also based on multi-head attention, but without skip-connections, batch normalization and the feed-forward sublayer for the concern of efficiency. (Bresson and Laurent 2021) is another Transformer-based TSP solver with slightly different customization in the decoder.

An alternative encoding is to treat the input cities as a complete graph, since each pair of cities is reachable. In (Yu, Yu, and Gu 2019), a graph embedding approach called `structure2vec` (Dai, Dai, and Song 2016; Khalil et al. 2017) is applied on the input nodes. Recently, graph convolutional encoder is used in (Luo et al. 2022) to extract hierarchical features. Its decoder scheme consists of more than one multi-head attention decoder with identical structures but unshared parameters.

### Learning-to-Improve Methods

The learning-to-improve methods combine RL with heuristic methods for more effective search space exploration. Given an initial solution, the RL agent learns to select better neighborhood operation to iteratively improve the current best tour. In (Zheng et al. 2021), value-based RL approach is adopted to improve LKH heuristic. In (Fu, Qiu, and Zha 2021), LKH is applied for small neighborhood exploration, followed by RL-based Monte Carlo tree search (MCTS) to handle enlarged neighborhood. In (Xin et al. 2021), sparse graph network is proposed to create edge candidate sets, with the purpose of guiding the search space exploration in LKH. We also notice the combination with RL and heuristic algorithms in more complicated scheduling problems, such as capacitated vehicle routing problems (CVRP) (Lu, Zhang, and Yang 2020).

The research scope in this paper falls into the category of *learning-to-generate* and we will mainly compare with the state-of-the-art encoder-decoder models. It can be viewed as complementary to learning-to-improve methods because they can apply our model to generate initial solution for further improvement.

## Proposed Model

Our TSP solver is built on top of the mainstream encoder-decoder models, which are trained with reinforcement learning to eliminate the reliance on high-quality training labels. Our technical contributions are three-fold: 1) a progressive distillation framework combining curriculum learning to train the samples in increasing order of their difficulty and a novel distillation loss to transfer the knowledge from pre-trained small models to the large models; 2) an action mask based on Delaunay graph; and 3) an attention-based decoder with better tradeoff between efficiency and quality. Details of each component are explained in the following.

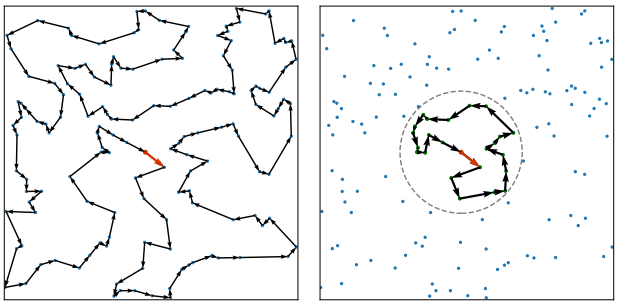
### Curriculum Learning

Curriculum learning (CL) has been shown to be effective to tackle tasks that are difficult to learn from scratch (Narvekar et al. 2020) and has been applied for neural combinatorial optimization in previous work (Lisicki, Afkanpour, and Taylor 2020). To apply CL, our strategy is to use problem size (i.e., the number of input cities) as difficulty indicator and develop a curriculum of training samples with increasing number of cities. In our setting of the curriculum schedule to train samples from size $m$ to $n$ ($n > m$), we adopt a parameter $C$ to determine the number of training epochs. Thereafter, in the $i$-th epoch, we will use TSP instances with size $m + i\lfloor\frac{n-m}{C}\rfloor$ for model training. For example, suppose we start from TSP50 samples and our final goal is to train TSP250, i.e., $m = 50$ and $n = 250$. If we set $C = 4$, we use samples of TSP100 in the first training epoch. When the model parameters are updated, we will use training samples of TSP150 in the next epoch. The process repeats until TSP250 samples are trained in the fourth epoch. Let $N$ ($N > C$) be the hyperparameter to control the total number of training epochs. For the remaining $N-C$ epochs, the model will continue to be trained with TSP samples of size $n$ (i.e., 250 in this example).

### Distillation Loss

Before we formally present the definition of distillation loss, we start with a toy example to illustrate the motivation. Figure 1a shows an instance of TSP150 with 150 points

generated by uniform distribution in the 2D space $[0, 1]^2$, as well as its optimal visit path $P_g$ derived from an exact solver. In the setting of Figure 1b, we select one of the cities $p_i$ and build a local circular window centered at $p_i$. The size of the circular window is determined by a ratio parameter $r$ to determine the number of cities in the window. If we set $r = 0.4$, there are 60 cities in the circle, which constitute a small-scale TSP instance. We mark those preceding $p_i$ in the optimal path $P_g$ as visited. Afterwards, we run the exact TSP solver for the small-scale instance to obtain its optimal solution $P_s$ for the unvisited cities within the local window. We observe that there is high chance (128 out of 150 points in this example) of selecting the same next city for $p_i$ in the optimal solutions $P_g$ and $P_s$. In other words, we can consider that the cities far away from $p_i$ play small effect when determining the optimal next city to visit.



(a) The global optimal solution. (b) The local optimal solution.

Figure 1: Consistency of optimal action selection from global view and local view.

Inspired by the observation, we can use an RL agent that has been well trained in small-scale instances as the teacher network to guide the training of another RL agent on large-scale TSP problems, which is viewed as the student network. Even though traditional knowledge distillation refers to knowledge transfer from a big teacher network to a small network, we still call our large-scale TSP model as student network because it can benefit from the guidance of the small teacher network when determining the next city to visit in a local circular area.

To transfer the knowledge learned from the teacher network to the student network, we follow the conventional way of introducing a distillation loss $\mathcal{L}_{kd}$ that minimizes the difference between the logits produced by these two networks. Let $p_\theta(u_t|s_t)$ be the policy learnt by the the student network for large-scale TSP instances and $p_{\theta'}(u'_t|s'_t)$ be the policy learnt by teacher network for small-scale instances. Here, $t$ refers to $t$-th step of decoding, $s_t$ is the state and $u_t$ is the logits produced by the policy network. In our definition of $\mathcal{L}_{kd}$, we intend to minimize the difference between these two policies. To ensure that they are comparable, we need to project $p_\theta(u_t|s_t)$ to the same action space with the teacher network. This is done by introducing a mask $M_{local}$ to remove cities outside the local region (as shown in Figure 1b) and normalize the probability distribution for the remaining cities:

$$p_\theta^{local}(u_t|s_t) = \texttt{softmax}(p_\theta(u_t|s_t) \cdot M_{local})$$

With the derived $p_\theta^{local}(u_t|s_t)$, we use KL divergence to measure the discrepancy between the policies generated by teacher network and student network:

$$\mathcal{L}_{kd} = \sum_{t=1}^{T} D_{KL}(p_{\theta'}(u'_t|s'_t) \,||\, p_\theta^{local}(u_t|s_t))$$

Finally, the distillation loss $\mathcal{L}_{kd}$ is integrated with original RL policy training loss and trained by REINFORCE (Williams 1992) gradient estimator with baseline $b(s)$.

$$\mathcal{L}(\theta|s) = \boldsymbol{E}_{p_\theta(\pi|s)}\Big[(\mathcal{L}(\pi) - b(s)) \log p_\theta(\pi|s) + \beta \mathcal{L}_{kd}\Big]$$

## Progressive Distillation Framework

The integration of curriculum learning and knowledge distillation constitutes our progressive distillation framework. On one hand, the curriculum learning strategy progressively trains the model with more and more difficult samples, resulting in multiple snapshots of trained models. On the other hand, these trained models in previous snapshots can be further used by the teacher network to guide the training of the current model.

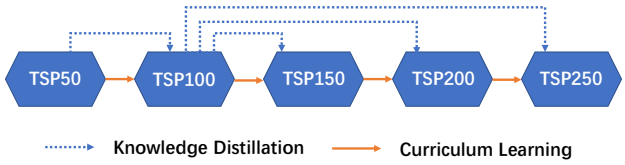



Figure 2: An example of progressive distillation from TSP50 to TSP250.

To facilitate understanding, we present an example in Figure 2 to explain the idea of progressive distillation. Suppose we start from a model fully trained on TSP50 instances and our goal is to train a new model to handle TSP250. We set $N = 10$, $C = 4$ and the ratio for the local window size to be 0.4. In the first epoch, we train the model using samples of TSP100. In the meanwhile, the number of cities in the local window is $0.4 * 100 = 40$ and we apply the pre-trained TSP50 model as the teacher network for knowledge transfer. The generated distillation loss is incorporated in the training objective for parameter update. In the second epoch for TSP150, we repeat the process and select TSP100 instead of TSP50 as the teacher network. This is because the number of cities in the local window is $0.4 * 150 = 60$ and we pick its closest snapshot with larger problem size. After that, we continue the training with TSP200 and TSP250, respectively. Finally, when we finish the curriculum learning process, we will train the model with only TSP250 samples for another $10 - 4 = 6$ epochs, using TSP100 as the teacher network.

The pseudo code of our model training based on progressive distillation is depicted in Algorithm 1. The pre-trained model $\theta_0$ from an existing approach is used for

Algorithm 1: Model training by progressive distillation

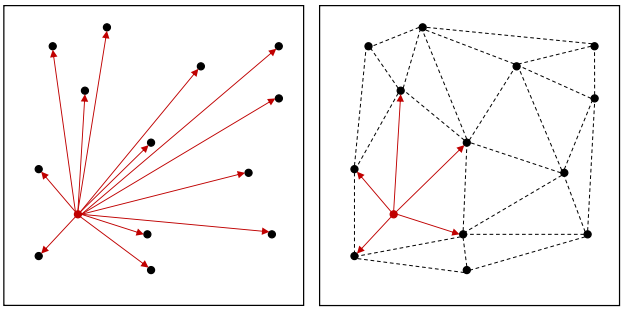
**Input**: total training epochs $N$, curriculum learning epochs $C$, steps per epoch $T$, starting TSP size $m$, ending TSP size $n$, distillation learning rate $\beta$, pre-trained model $\theta_0$, local window size ratio $r$

Initialize $\theta \leftarrow \theta_0$
**for** epoch $\leftarrow 1, \ldots, N$ **do**
  $p\_size \leftarrow n$
  **if** $epoch \leq C$ **then**
    $p\_size \leftarrow m + epoch\lfloor \frac{n-m}{C} \rfloor$
  **end if**
  Select teacher network based on $r * p\_size$
  **for** step = 1, ..., T **do**
    generate batch of training samples with $p\_size$ cities
    $\mathcal{L}_{kd} \leftarrow 0$
    **for** each decoding step $t$ **do**
      Generate city mask $M_{local}$ based on state $s_t$
      $p_\theta^{local}(u_t|s_t) = \texttt{softmax}(p_\theta(u_t|s_t) \cdot M_{local})$
      $\mathcal{L}_{kd} \leftarrow \mathcal{L}_{kd} + D_{KL}(p_{\theta'}(u_t'|s_t') \,||\, p_\theta^{local}(u_t|s_t))$
    **end for**
    $\bigtriangledown\mathcal{L} \leftarrow (L(\pi) - b(s)) \bigtriangledown_\theta \log p_\theta(\pi) + \beta \bigtriangledown_\theta \mathcal{L}_{kd}$
    $\theta \leftarrow Adam(\theta, \bigtriangledown\mathcal{L})$
  **end for**
**end for**

model parameter initialization. In the first $C$ samples, the model is trained with a curriculum of increasing size of TSP samples, with additional knowledge distillation from previous snapshots. In the last $N - C$ epochs, the model is trained with only samples of size $n$ and the distillation loss for action guidance in the local window is still applied.

**Delaunay Graph-based Action Mask** In previous TSP solvers, the cities are selected step by step and their action space consists of all the unvisited cities. However, we observe that cities far away are unlikely to generate the optimal solution and can be excluded from the search space. In other words, previous approaches can be considered as using complete graph to model action space, with an additional mask to exclude cities that have been visited. In contrast, we intend to build a graph on the data points to capture their spatial proximity. The action mask is defined in a way that only the neighbors in the graph will be the next city to visit (as shown in Figure 3). We choose Delaunay graph because it has several desirable properties to capture proximity relationship. For each point $p_i$, there exists an edge $(p_i, p_j)$ in $G_P$ such that $p_j$ is the nearest neighbor of $p_i$. This implies that the closest neighbor graph is in fact a subset of Delaunay graph. Another property is that no point in $P$ is inside the circumference of any triangle in $G_P$. Such no-crossing-edge property could be useful for TSP scheduling.

**Input Encoding** Unlike previous works, we propose a new encoding strategy that relies on Delaunay graph. As mentioned in (Kool, van Hoof, and Welling 2019), the attention mechanism can be interpreted as a weighted message passing algorithm between nodes in a graph. We use the adjacency relationship in the Delaunay graph to calculate the compatibility $u_{ij}$ for nodes $i$ and $j$:



(a) Complete Graph (b) Delaunay Graph

Figure 3: Motivation of using Delaunay graph.

$$u_{ij}^{(t)} = \begin{cases} \frac{q_i^{(t)} \cdot k_j^{(t)}}{\sqrt{d_k}}, & \text{if node } i \text{ is adjacent to } j \\ -\infty, & \text{otherwise} \end{cases} \quad (1)$$

From $u_{ij}^{(t)}$, we can compute attention weight

$$a_{ij}^{(t)} = \text{softmax}(u_{ij}^{(t)}) \quad (2)$$

Finally, the vector $h_i^{(t)}$ received by node $i$ is the combination of messages from neighbors $v_j$.

$$h_i^{(t)} = \sum_j a_{ij}^{(t)} v_i^{(t)} \quad (3)$$

**Attention-Based Decoder** Attention-based decoder has been frequently used in previous RL-based models to improve the accuracy of next city prediction. At step $t$, the decoder outputs city $\pi_t$ based on the embeddings from the encoder and previous cities $\pi_{t'}$ with $t' < t$. In (Kool, van Hoof, and Welling 2019), the Transformer-style multi-head attention was adopted. To avoid interaction with all the previous outputs, its query vector $q_c$ is defined on top of a context vector $h_c$:

$$q_c = W^Q h_c \quad (4)$$

$$h_c^{(t)} = [\overline{h}, h_{\pi_{t-1}}, h_{\pi_1}] \quad (5)$$

Here, $\overline{h}$ is the mean of embeddings for all the cities, $h_{\pi_{t-1}}$ is the output from the previous decoded city and $h_{\pi_1}$ is the output of the first decoded city. In (Bresson and Laurent 2021), an additional layer of self-attention is used and all the outputs from previous steps $h_{\pi_1}, h_{\pi_2}, \ldots, h_{\pi_{t-1}}$ will be taken into account. Furthermore, its decoder is required to query the next possible city among the non-visited cities using a query-attention layer. Therefore, its computation complexity is $O(n)$, in contrast to $O(1)$ in (Kool, van Hoof, and Welling 2019).

To obtain a better trade-off between computation overhead and training effectiveness, we propose a new type of attention-based decoder which runs much faster than (Bresson and Laurent 2021) and is more effective than (Kool, van Hoof, and Welling 2019). The idea is replace $h_c$ used in (Kool, van Hoof, and Welling 2019) with a more

| Method | TSP150 | | | TSP300 | | | TSP500 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Length | Gap | Time | Length | Gap | Time | Length | Gap | Time |
| PtrNet-RL | 12.907 | 37.63% | 3.96 | 22.428 | 72.81% | 12.72 | 34.631 | 108.86% | 22.92 |
| GCN | 12.201 | 30.10% | 0.90 | 20.646 | 59.08% | 5.40 | 29.029 | 75.07% | 17.28 |
| Transformer-1 | 10.390 | 10.79% | 0.24 | 16.624 | 28.09% | 0.78 | 22.560 | 36.04% | 1.92 |
| Transformer-2 | 9.857 | 5.11% | 0.66 | 15.811 | 21.83% | 1.68 | 22.315 | 34.58% | 4.02 |
| GCE-MAD | 10.225 | 9.03% | 0.36 | 16.459 | 26.82% | 1.44 | 23.047 | 39.00% | 3.78 |
| PDAM | **9.830** | **4.84%** | 0.30 | **14.641** | **12.81%** | 0.84 | **19.727** | **18.97%** | 2.04 |

Table 1: Comparison with learning-to-generate models on TSP150, TSP300 and TSP500. The average inference time is in the unit of millisecond.

informative context vector. We observe that $\overline{h}$ is in fact a static vector during decoding and cannot capture the dynamic context at different step $t$. Our strategy is to decompose it into two dynamic vectors $[h_v^{(t)}, h_u^{(t)}]$, with $h_v^{(t)}$ capturing the context from decoded cities and $h_u^{(t)}$ aggregating the embeddings for the non-visited cities. To avoid examining all the previous cities $\pi_1, \pi_2, \ldots, \pi_{t-1}$, we introduce a forget gate and define $h_v^{(t)}$ in a recursive manner:

$$h_v^{(t)} = z h_v^{(t-1)} + (1-z) u^{(t-1)} \quad (6)$$

$$z = \phi(W^z h_z) \quad (7)$$

$$h_z = [h_v^{(t-1)}, h_u^{(t)}, h_{\pi_{t-1}}, h_{\pi_1}] \quad (8)$$

Here, $u^{(t-1)}$ is the output of multi-head attention calculated by $\texttt{softmax}(\frac{QK^T}{\sqrt{d_k}})V$. For the definition of $h_u^{(t)}$, it is explicitly represented as the mean of embeddings for all the legal cities that are non-visited and not excluded by the Delaunay graph-based action mask. More details of implementation are provided in the technical report[1].

## Experimental Study

In this section, we conduct extensive experiments for performance evaluation. We denote our progressive distillation method as **PDAM**. Since it belongs to *learning-to-generate*, we select multiple representative approaches in this category as our comparison baselines, including **GCN** (Joshi, Laurent, and Bresson 2019) as a supervised model with graph embedding, **PtrNet-RL** (Bello et al. 2017) as an encoder-decoder framework with RL, **Transformer-1** (Kool, van Hoof, and Welling 2019) and **Transformer-2** (Bresson and Laurent 2021) as two Transformer-based approaches, and **GCE-MAD**(Luo et al. 2022) that uses graph convolutional encoder and multiple Transformer-based decoders.

Since PDAM can instantly return a satisfactory result, we also validate its usefulness as an initial solution generator that is complementary to the state-of-the-art TSP solvers, including LKH family (Helsgaun 2000, 2009, 2017) as the most competitive heuristic algorithm, and LKH3-RL (Fu, Qiu, and Zha 2021) and NeuroLKH (Xin et al. 2021) as the representative learning-to-improve methods. In terms of the exact solver, we follow (Zheng et al. 2021) to run Concorde[2] and return exact solutions.

For datasets, we follow previous learning-to-generate models to use uniform distribution in two-dimensional space $[0, 1]^2$. In addition, we examine the performance in the Euclidean TSP instances in the TSPLIB repository[3]. Due to space limit, the results have been documented in the technical report .

As to hyper-parameter setting, we set $N = 100$, $C = 60$, $r = 0.4$, $\beta = 0.1$ and $T = 2000$ for the training framework based on progressive distillation. The Transformer has 6-layers encoder and 1-layer decoder, both with 128 hidden units in each layer.For batch size, we set 512 for TSP150, 128 for TSP300 and 32 for TSP500. This is because training TSP500 is expensive in memory consumption and we have to reduce the batch size to avoid out-of-memory error. In the implementation of decoder, we simply apply greedy search and select the next city with the highest probability for each decoding step. All the experiments are conducted on a single GPU (NVIDIA Tesla V100 with 32GB memory).

### Comparison with Learning-to-Generate Models

In the first experiment, we compare PDAM with existing learning-to-generate models and report the tour length, gap to the optimality, and inference time in Table 1 on three scales, including TSP150, TSP300 and TSP500. For each scale, we randomly generate $1,000$ query instances in the Euclidean space and report the average tour length and inference time per instance. Among the learning-to-generate models, PtrNet-RL and GCN exhibit poor performance in terms of both inference time and solution quality. Their running time grows dramatically with the increasing of problem size because the RNN decoding in PtrNet-RL is expensive and GCN requires beam search for tour inference. Their performance gap is also widened from TSP150 to TSP500 because their model training becomes more difficult to get converged. GCN relies on supervised learning and requires a considerable number of training labels. PtrNet-RL adopts RNN-based network and is less effective than the Transformer-based models. We also note that Transformer-1 achieves similar performance to GCE-MAD. Transformer-2 is the most effective among the comparison models.

[1] https://www.docdroid.net/9bhLAIS/report-pdf

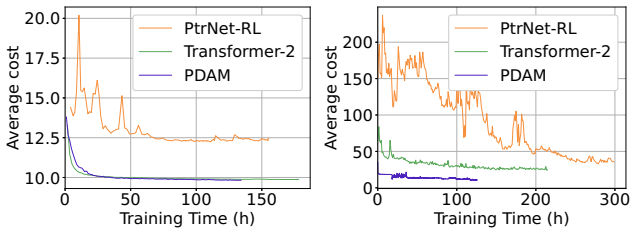
[2] http://www.math.uwaterloo.ca/tsp/concorde/index.html

[3] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

| Method | TSP300 | | | TSP500 | | | TSP1000 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Length | S-Rate | Time | Length | S-Rate | Time | Length | S-Rate | Time |
| Concorde (exact) | 12.978 | 100% | 3.66 | 16.581 | 100% | 18.72 | 23.131 | 100% | 190.56 |
| LKH-3 | 13.002 | 83.1% | 0.48 | 16.696 | 78.9% | 1.74 | 24.005 | 75.3% | 20.64 |
| LKH3-RL | 12.895 | 85.4% | 0.84 | 16.678 | 81.4% | 2.94 | 23.575 | 77.4% | 36.72 |
| NeuroLKH | 12.890 | 86.2% | 0.96 | 16.620 | 83.5% | 3.36 | 23.272 | 78.1% | 42.12 |
| PDAM + LKH3 | 12.998 | +0.7% | 0.48 | 16.678 | +1.3% | 1.77 | 23.870 | +0.8% | 21.14 |
| PDAM + LKH3-RL | 12.892 | +0.1% | 0.85 | 16.653 | +2.2% | 2.96 | 23.429 | +0.9% | 36.84 |
| PDAM + NeuroLKH | 12.887 | +0.7% | 0.96 | 16.613 | +0.9% | 3.42 | 23.259 | +0.7% | 42.78 |

Table 2: Compare strong TSP solvers with their counterparts of applying PDAM for initial solution generation. S-Rate refers to the success rate of achieving the optimal solution. The inference time is in the unit of second.

Compared with Transformer-2, our proposed PDAM derives significantly better solutions in large-scale TSP instances. The performance gap to optimality in TSP500 is reduced from 34.58% to 18.97% with even less inference time.




(a) TSP150 (b) TSP500

Figure 4: Training convergence.

## Model Training Convergence

In Figure 4, we investigate the pattern of training convergence for the RL-based learning-to-generate models. In TSP150, the convergence patterns of Transformer-2 and PDAM are smooth and close to each other. In contrast, PtrNet-RL shows dramatic fluctuation in the early training stage. When the problem size increases to TSP500, the training becomes much more challenging. We can see that it is difficult for PtrNet-RL to get converged. GCN shows an overall trend of convergence, but to an inferior local optimal. Our PDAM can steadily get converged in TSP500 and it is interesting to observe the stage-wise pattern for cost decline, because the model is trained with curriculum learning framework.

## Effectiveness as Initial Solution Generator

In this experiment, we validate the usefulness of PDAM by applying it as the initial solution generator for strong TSP solvers, including the heuristic LKH-3 and two learning-to-improve methods (LKH3-RL and NeuroLKH). From the results in Table 2, we can observe that the quality of the returned tours is improved from two perspectives. First, the average tour length can be further reduced. For example, PDAM+LKH3 reduces the tour length returned by LKH-3 from 24.005 to 23.870 in TSP1000. Second, we can observe that using PDAM can improve the success rate (abbreviated as S-Rate) of obtaining the optimal solution. Similar experimental results on selected real datasets from TSPLIB are reported in Table 3. Due to space limit, more details are presented in the supplementary material.

| Method | gr229 | gr431 | gr666 |
|---|---|---|---|
| LKH | 2/10 | 3/10 | 5/10 |
| LKH-RL | 7/10 | 7/10 | 9/10 |
| NeuroLKH | 5/10 | 8/10 | 9/10 |
| PDAM+NeuroLKH | 10/10 | 9/10 | 10/10 |

Table 3: Success rate of obtaining optimal solution in three real datasets gr229, gr431 and gr666 in TSPLIB.

## Analysis on Modal Adaptation

We also examine the adaptation of the learned models to tackle problems with larger scale. More specifically, we apply the models trained on TSP300 to run instances of larger scale, including TSP500, TSP1000 and TSP2000. The results are reported in Table 4, where LKH-3 is used as the baseline to estimate the performance gap. Compared with the two Transformer-based models, our PDAM achieves the best performance and the superiority is widened when the problem size increases. Such generality owes to the curriculum learning strategy to train the model with TSP instances in different scales. It's also interesting to observe that even though in previous experiment the fully trained model of Transformer-1 is inferior to Transformer-2 (as shown in Table 1), it demonstrates slightly better adaptation to handle larger-scale TSP instances, probably due to its simpler network structure.

## Ablation Study

In the ablation study, we assess the effect of the proposed techniques in PDAM, including curriculum learning framework, distillation loss for knowledge transfer, the graph based action mask and the attention-based decoder. Note that when the curriculum learning framework is discarded, distillation loss is still applicable. For example, to train TSP250, we can still use a fully trained TSP100 model as the teacher

| Method | TSP300 | | TSP500 | | TSP1000 | | TSP2000 | |
|---|---|---|---|---|---|---|---|---|
| | Length | Gap | Length | Gap | Length | Gap | Length | Gap |
| LKH-3 | 12.98 | 0.0% | 16.58 | 0.0% | 23.24 | 0.0% | 32.76 | 0.0% |
| Transformer-1 | 16.62 | 28.0% | 22.79 | 37.5% | 33.87 | 45.7% | 53.92 | 64.6% |
| Transformer-2 | 15.81 | 21.8% | 22.55 | 36.0% | 34.17 | 47.0% | 54.57 | 66.6% |
| GCE-MAD | 16.43 | 26.6% | 22.89 | 38.1% | 34.05 | 46.5% | 54.29 | 65.7% |
| PDAM | 14.64 | 12.8% | 20.51 | 23.7% | 29.85 | 28.4% | 47.18 | 44.0% |

Table 4: Models trained on TSP300 and tested on larger instances.

network to guide action selection in TSP250. In Table 5, we train these models with the same number of epochs on TSP150 instances and report the final solution quality for performance comparison. The results show that curriculum learning and distillation loss are two key components in PDAM, with considerable performance degradation when they are removed. The graph-based action mask also plays positive effect, reducing the average tour length from 9.89 to 9.83.

| Method | Length | Gap |
|---|---|---|
| Full model | 9.83 | 0% |
| w/o curriculum | 10.09 | 2.62% |
| w/o distillation loss | 9.97 | 1.45% |
| w/o action mask | 9.89 | 0.55% |

Table 5: Ablation study.

To assess the advantage of our proposed decoder, we replace it with alternative Transformer structures implemented in Transformer-1, Transformer-2 and GCE-MAD, respectively. The results in Table 6 show that our decoder achieves the best trade-off between training/inference efficiency and solution quality. It achieves comparable solutions with Transformer-2, but with only its half training/inference time. Even though GCE-MAD has multiple decoders, its efficiency is better than Transformer-2 because these decoders can run in parallel.

| Method | Length | Train time | Test time |
|---|---|---|---|
| Our decoder | 9.8302 | 113 h | 0.28 s |
| Transformer-1 | 9.9507 | 108 h | 0.27 s |
| Transformer-2 | 9.8268 | 281 h | 0.61 s |
| GCE-MAD | 9.9384 | 168 h | 0.57 s |

Table 6: Comparison with different decoders.

In Figure 5, we pay a closer attention to the advantages brought by curriculum learning, and reveal the performance improvement when the samples are trained in increasing order of problem size. When we start from a pre-trained TSP10 model to train TSP150, we split the training curve into four stages: 1) from TSP10 to TSP50; 2) from TSP50 to TSP100; 3) from TSP100 to TSP150 and 4) the final stage with only TSP150 samples. We compare with the baseline model without curriculum learning, which directly trains a TSP150 model from scratch using training samples of size 150. As shown in the figure, our model demonstrates much faster convergence pattern in the first 20 hours of training. It is also interesting to find that before our curriculum learning covers TSP150 training sample (i.e. before reaching stage 4), it has already achieved better performance than the baseline. This finding verifies the effectiveness of the curriculum learning framework. We also compare with another transfer learning strategy which trains TSP150 with a warm start from a fully trained TSP50 model. The observation is that the training of TSP150 can benefit from the warm start trick and converges to a better solution than the baseline which is trained from scratch. Nevertheless, the warm start training strategy is still inferior to our curriculum learning framework.

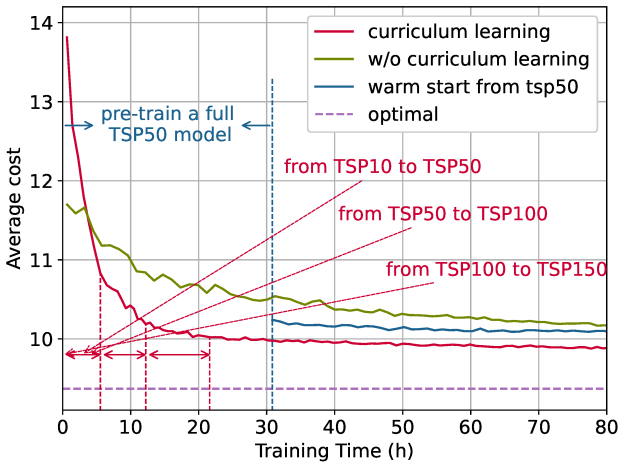



Figure 5: Effect of curriculum learning.

## Conclusion

In this paper, we proposed a new progressive distillation framework based on curriculum learning and knowledge distillation to tackle the high barrier of training large-scale TSPs. We also proposed a Delaunary graph based action mask and a new type of attention-based decoder for performance improvement. Experimental results validated the superiority of our approach over its competitors in this category. We also validate its advantage when applied as an initial solution generator for existing TSP solvers.

## Acknowledgements

The work is supported by the National Key Research and Development Project of China (2022YFF0902000).

## References

Bektas, T. 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3): 209 – 219.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *ICLR (Workshop)*. OpenReview.net.

Bresson, X.; and Laurent, T. 2021. The Transformer Network for the Traveling Salesman Problem. *CoRR*, abs/2103.03012.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML*, 2702–2711.

Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; and Rousseau, L. 2018. Learning Heuristics for the TSP by Policy Gradient. In *CPAIOR*, volume 10848 of *Lecture Notes in Computer Science*, 170–181. Springer.

Fu, Z.; Qiu, K.; and Zha, H. 2021. Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. In *AAAI*, 7474–7482. AAAI Press.

Helsgaun, K. 2000. An effective implementation of the Lin¨CKernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1): 106–130.

Helsgaun, K. 2009. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Math. Program. Comput.*, 1(2-3): 119–163.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*.

Joshi, C. K.; Laurent, T.; and Bresson, X. 2019. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. *CoRR*, abs/1906.01227.

Khalil, E. B.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *NIPS*, 6351–6361.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *ICLR*.

Laporte, G. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2): 231 – 247.

Lisicki, M.; Afkanpour, A.; and Taylor, G. W. 2020. Evaluating Curriculum Learning Strategies in Neural Combinatorial Optimization. *CoRR*, abs/2011.06188.

Lu, H.; Zhang, X.; and Yang, S. 2020. A Learning-based Iterative Method for Solving Vehicle Routing Problems. In *ICLR*. OpenReview.net.

Luo, J.; Li, C.; Fan, Q.; and Liu, Y. 2022. A graph convolutional encoder and multi-head attention decoder network for TSP via reinforcement learning. *Engineering Applications of Artificial Intelligence*, 112: 104848.

Ma, Q.; Ge, S.; He, D.; Thaker, D.; and Drori, I. 2019. Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning. *CoRR*, abs/1911.04936.

Narvekar, S.; Peng, B.; Leonetti, M.; Sinapov, J.; Taylor, M. E.; and Stone, P. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *J. Mach. Learn. Res.*, 21: 181:1–181:50.

Nazari, M.; Oroojlooy, A.; Snyder, L. V.; and Takác, M. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. In *NeurIPS*, 9861–9871.

Peng, Y.; Choi, B.; and Xu, J. 2021. Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art. *Data Sci. Eng.*, 6(2): 119–141.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NIPS*, 6000–6010.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In *NIPS*, 2692–2700.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8: 229–256.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021. NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. *Advances in Neural Information Processing Systems*, 34.

Yu, J. J. Q.; Yu, W.; and Gu, J. 2019. Online Vehicle Routing With Neural Combinatorial Optimization and Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*, 1–12.

Zheng, J.; He, K.; Zhou, J.; Jin, Y.; and Li, C. 2021. Combining Reinforcement Learning with Lin-Kernighan-Helsgaun Algorithm for the Traveling Salesman Problem. In *AAAI*, 12445–12452. AAAI Press.