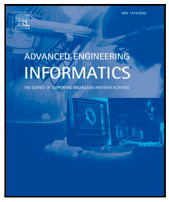
Advanced Engineering Informatics 56 (2023) 102005

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Full length article

Shared dynamics learning for large-scale traveling salesman problem

Yunqiu Xu [a], Meng Fang [b], Ling Chen [a], Yali Du [c], Gangyan Xu [d,*], Chengqi Zhang [a]

[a] School of Computer Science, The University of Technology Sydney, Sydney, Australia
[b] Department of Computer Science, University of Liverpool, UK
[c] Department of Informatics, Faculty of Natural, Mathematical & Engineering Sciences, King's College London, London, UK
[d] Department of Aeronautical and Aviation Engineering, Faculty of Engineering, Hong Kong Polytechnic University, Hong Kong

ARTICLE INFO

Keywords:
Traveling salesman problem
Combinatorial optimization
Multi-task learning
Reinforcement learning
Zero-shot learning
Few-shot learning

ABSTRACT

In this work, we study generalization in reinforcement learning for traveling salesman problem (TSP). While efforts have been made for designing deep reinforcement learning-based solvers to achieve near optimal results in small tasks, it is still an open problem to apply such solvers to larger-scale tasks by retaining performance. In this research, we learn the shared dynamics in TSP environments based on multi-task learning, which can be generalized to new tasks. To accurately estimate such dynamics, we consider leveraging the node visitation information. Besides designing RL-based models to attentively aggregate the visitation information during decision making, we propose a scheduled data utilization strategy to stabilize learning with various problem sizes. The experimental result shows that our model achieves improved generalizability for unseen larger TSPs in both zero-shot and few-shot settings.

1. Introduction

The Traveling Salesman Problem (TSP) is regarded as one of the most fundamental combinatorial optimization problems that has been studied for many decades in multiple areas such as computer science, operations research and transportation [1,2]. Given a set of cities and the distance between each pair, the goal of TSP is to find the shortest possible route to visit each city exactly once and return to the origin city [3]. TSP is known as an NP-hard problem — the time required for finding the optimal solution grows exorbitantly with the problem size. Traditionally, the methods for solving TSP could be categorized as the exact methods and the approximate methods. By sacrificing the computing time, the exact approaches guarantee to obtain the optimal solution in small-size problems [4–6]. The approximate approaches try to balance the optimality and the computational efficiency by introducing various meta-heuristics [7–10]. These meta-heuristics methods could well cope with complex constraints, relatively large-size problems, and generate high quality solutions efficiently, which dominant the applications in different scenarios, such as bus route design [11], patrol routes design [12], and e-commerce logistics [13,14]. However, these methods require expert knowledge to design suitable heuristics, and its efficiency and solution quality will decrease dramatically along with the increase of problem size and complexity. They limit the applications in highly dynamic, time-sensitive, and large-scale problems, such as online ride-sharing problems, emergency responses, and autonomous systems.

The past few years have witnessed the premises of designing the TSP solvers with data-driven machine learning techniques [15]. The Deep Reinforcement Learning (DRL), in particular, becomes attractive as it reduces the dependency of expert knowledge by learning from interaction with data, removes the requirement of optimal labels by directly optimizing the task objective [16,17], and allows solving large-scale combinatorial optimization problems in near real-time with high quality solutions [18,19]. Typical Reinforcement Learning (RL) based TSP solvers [20–22] follow the encoder–decoder framework. At the beginning of each episode, the nodes will be first encoded by an encoder to get the node embeddings, and the overall context embedding. Then for each decoding step, a decoder selects one node conditioned on the context embedding and dynamic information, such as the last chosen node. By integrating the framework with search heuristics [23,24], the solvers can generate near-optimal solutions in relatively small TSPs (e.g., TSP20, TSP50 and TSP100), with faster computing speed than traditional solvers. However, it is still an open problem to efficiently solve large-scale TSPs, especially with unknown scenarios. Due to huge computation time and GPU memory requirement, it is intractable to train a model for such problems from scratch. Instead, training a model on small TSPs first and then transferring it to unseen larger problems seems more applicable, although considering such zero-shot or few-shot generalization requires rethinking both experimental and model architectural settings [25].

* Corresponding author.
E-mail address: gangyan.xu@polyu.edu.hk (G. Xu).

https://doi.org/10.1016/j.aei.2023.102005
Received 27 November 2022; Received in revised form 1 April 2023; Accepted 4 May 2023
Available online 24 May 2023
1474-0346/

In this research, we aim at improving the generalizability of RL-based model for larger-scale TSPs, while keeping its flexibility to be applied beyond TSP. Drawing inspiration from multi-task learning [26,27], we treat TSPs with different problem sizes as tasks with similar dynamics and objectives. Then, we try to capture the shared dynamics by designing attention-based models with node visitation information during the solution decoding process. Besides, motivated by the idea that similar tasks should be learnt together to better capture shared knowledge [28,29], we propose a data utilization strategy to schedule training tasks according to their problem sizes.

We conduct experiments on both zero-shot learning and few-shot learning settings. The results indicate that our model achieves enhanced generalizability in unseen larger TSPs. When provided with a few fine-tuning samples, our model could be quickly adapted to the target problem size. Additionally, our model shows consistent advantage against the baseline model with smaller fine-tuning batch size, making it more applicable to larger-scale problems.

Our contribution includes four aspects. Firstly, we are the first effort that studies generalization problem in TSP based on multi-task learning by identifying the objective as modeling shared dynamics. Secondly, we bring model-level improvement by attentively considering node visitation information during decoding. Thirdly, we bring training-level improvement by introducing scheduled data utilization strategy to stabilize learning under multi-task setting. Fourthly, we empirically validate our proposed model in both zero-shot generalization and few-shot generalization experiments, achieving favorable results compared with strong baselines.

The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 formally defines the problem. Section 4 demonstrates our methodology. Section 5 details the experiment settings. Section 6 discusses the results. Finally, Section 7 concludes the whole paper and identifies the future directions.

2. Literature review

The methods for solving TSPs in various scenarios have been extensively studied. The traditional methods can be categorized as the exact approaches and the approximate approaches. The exact approaches specialized for TSP, such as branch-and-Cut [30] and column generation [31], guarantee the optimality of the solutions. Nevertheless, these methods have high computational complexity, making them impractical to solve large-scale TSPs. In order to make larger problem tractable, the approximate approaches introduce various heuristics solution methods [32–34]. While the traditional methods require expert knowledge to design the heuristics, we propose to obtain a TSP solver through learning from trial and error, mitigating the demand for the handcrafting and prior knowledge.

Before the prosperity of deep neural-network-based solvers, Hopfield and Tank [35] applied the Hopfield network to solve small-scale TSPs. In terms of RL-based solvers, some works formulated TSP as sequential decision making problem, and combined RL with genetic algorithms [36–40]. Vinyals et al. [16] introduced PtrNet, in which the nodes will first be encoded as node embeddings, then during each decoding step, a decoder computes attention value for each node, and selects the one with highest attention value. Since this framework is optimized via supervised learning, it is challenging to obtain adequate optimal solutions as labels, especially for large TSP instances. To tackle this challenge, the Neural Combinatorial Optimization model (NCO) [17] extended PtrNet to deep reinforcement learning by defining the negative of the total tour length as the reward. Some recent work [23,41] designed the encoder or decoder with the transformer architecture, which has proven its advantage over the Recurrent Neural Networks (RNNs) in a broad range of language processing and sequential processing tasks [16,42]. Some work utilized the graph neural networks (GNNs) [43–45] and the graph embedding techniques [46], as the nodes of a TSP instance could be treated as a graph. For example, through designing a hierarchical framework upon the PtrNet with a GNN-based encoder, the Graph Pointer Networks (GPN) [47] addressed the generalizability problem in larger-scale TSP, while the computation efficiency is hindered by re-encoding the nodes per decoding step. [48] transformed the input graph as linear graph before encoding, achieving linear computation complexity consequently. Our work is associated with the graph representation learning as well, since the node visitation information can be represented as the sub-graphs of visited and unvisited nodes. Our model thus complements these work, or recent advances of the GNNs [49].

This work inherits from MARM [50], which is our initial study about modeling the dynamics in solving the vehicle routing problems. However, this work differs from MARM in following three aspects. Firstly, in terms of the motivation, this work aims at addressing the generalization problem towards larger-scale TSPs, while in MARM, the generalization across different problem sizes is only mentioned slightly, and no specific solution is provided. Secondly, in terms of the model architecture, while MRAM only considered one type of dynamics and left other types as a future direction, this work extends MRAM by investigating three types of dynamics. Thirdly, in terms of the training method, this work proposes a novel training strategy to facilitate training with various problem sizes, while MRAM only considered the training set with a same problem size.

Apart from the encoder–decoder framework, some efforts have been made to integrate the DRL techniques with heuristics-based approaches [51–53]. Since these work also contain problem encoding and state representation construction, our model can be adjusted as a module for them to improve problem size-invariant generalizability. Although we focus on TSP in this work, our model can be extended to other routing problems such as VRP [20], TSP with sliding window [54] and multi-objective setting [55,56].

Through controlled experiments, Joshi et al. [25] first systematically identified the generalization problem in learning-based TSP solvers. We extend their work to improve the generalizability by modifying both model architecture and training strategy. Our work is complementary to [47,48], which also considered generalizing to larger problem sizes. Similar to our work, AM-D [57] also modeled visitation dynamics for CVRP tasks. However, there are two major differences. First, we focus on improving generalizability in TSP (i.e., training on small problems then transferring to larger problems), while in AM-D both the training and testing problems are of same size. Secondly, the node representation in AM-D requires to be rebuilt after finding a partial solution, while for TSP and other routing problems without partial solution, it is unfavorable to encode multiple times [20]. We instead modify the solution decoding process based on obtained node embeddings, avoiding node re-encoding.

Besides TSP related works, another stream that related to our paper is Multi-task reinforcement learning, which focuses on learning shared representation among tasks [58]. Existing work include policy distillation [59–62], actors/learner framework [28,63], learning transferable knowledge [64–66] and learning hidden parameters over MDPs [67,68]. Our work is related to the last two in terms that we aim to learn shared dynamics that can be generalized to new tasks. We also consider the similarity between tasks [29].

3. Problem definition

We follow Joshi et al. [25] to define the end-to-end neural combinatorial optimization pipeline as five-stages: problem definition, input encoding, solution decoding, solution search and policy learning. We center on the 2D-Euclidean TSP, and denote it as TSP for short hereafter. Each problem instance consists of a set of N nodes $s = \{x_1, x_2, \dots, x_i, \dots, x_N\} \in R^2$, in which a node x_i is specified by 2-D coordinates. s can also be represented as a complete graph with a connection probability between any pair of two nodes. The output of a problem instance is defined as a permutation of N node indices $a = (a_1, a_2, \dots, a_i, \dots, a_N)$,

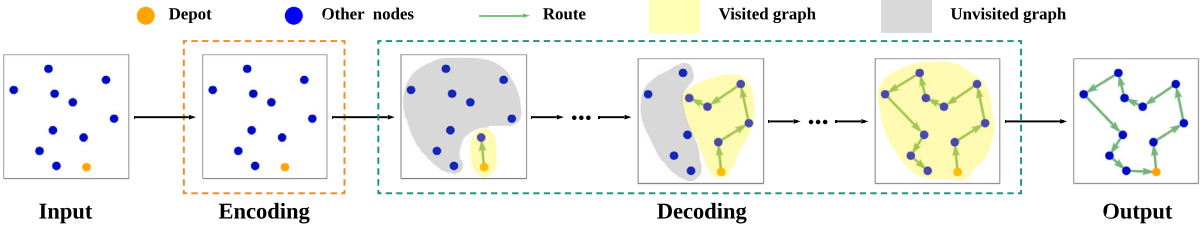


Fig. 1. The RL framework for TSP. Before an episode, the agent first encodes all nodes as the node embeddings, which will be fixed throughout this episode. In each decoding time step, the agent selects an unvisited node. The final output is a permutation of nodes, denoting the solution route of this problem instance. In this work we aim to better estimating the dynamics by modeling visitation information as sub-graphs (the shadowed parts).

in which a_i is the index of the ith node. The objective is to minimize the total tour length while visiting each node once (twice for the depot), which could be formulated as:

$$L(a|s) = \|x_{a_N} - x_{a_1}\|_2 + \sum_{i=1}^{N-1} \|x_{a_i} - x_{a_{i+1}}\|_2 \tag{1}$$

where the depot x_{a_1} is an arbitrary node.

Fig. 1 displays the general reinforcement learning framework for solving the TSP. For a problem instance $\mathcal{A}$ consisting of N nodes, we define an RL episode as two parts: the encoding process at the beginning of the episode, and the N-timestep decoding process to obtain the node permutation a as the output. During the encoding process, the agent will encode the node as the node embeddings. Then these node embeddings will remain fixed throughout this episode. The decoding process begins from an arbitrary node — for each decoding time step t, the agent selects an action a_t, which denotes the index of an unvisited node $a_t \in \mathcal{A} - \{a_1, a_2, \dots, a_{t-1}\}$. After visiting all the nodes exactly once, the agent will obtain the final permutation a through returning to the starting node. We define the state s_t as a combination of the node embeddings as well as the environmental dynamics, such as the last chosen node which changes over time. We define the reward as the negative of the total tour length $-L(a|s)$, and assign it episodically. The agent's learning objective lies in learning an optimal policy parameterized by θ^* to maximize the episodic reward:

$$p_{\theta^*}(a|s) = \prod_{t=1}^{N} p_{\theta^*}(a_t|s_t, a_{1:t-1}), \qquad s.t., a_t \in \mathcal{A} - \{a_1, a_2, \dots, a_{t-1}\} \tag{2}$$

While previous work have achieved promising performance in TSP with fixed problem size up to 100 nodes [17,20,21], our goal is to extend them to be invariant of problem sizes, especially to handle large TSPs, where training an RL model from scratch might be intractable. We cast the generalization problem as transfer learning and multi-task learning. We take TSP instances with same problem size as data from the same task, and treat tasks with small problem sizes as source tasks, while those with larger problem sizes as target tasks. These tasks share same goal and similar dynamics. Trained on source tasks, the model will then be adapted to target tasks either directly (zero shot), or with a few fine-tuning samples (few shot).

4. Methodology

4.1. Attention model with node visitation

We design our model upon the Attention Model (AM) [21], which is a strong baseline with comparable performance, fast running speed and high extensibility. The encoder of AM is similar to the transformer architecture except two major differences: (1) the positional encoding operation is removed from the encoder, as the input set of TSP is invariant of the order; and (2) the layer normalization operation is replaced by the batch normalization, which has been proven to be more effective for TSP. From the perspective of graph learning, the encoder could also be viewed as the Graph Attention Networks (GATs) [69] over a fully-connected graph.

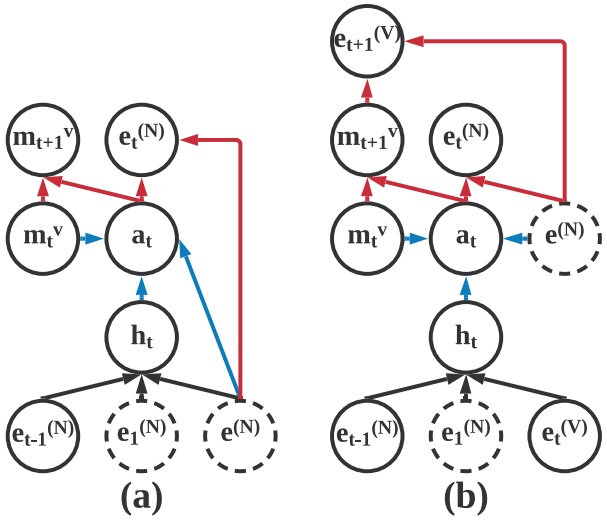


Fig. 2. The decoding process of (a) AM and (b) our model. The dashed variables are fixed throughout the episode that they cannot reflect transition dynamics. The solid variables change per time step. The black arrows denote the process of computing context embedding. The blue arrows denote the process of action scoring. The red arrows denote the process of updating variables for next time step.

For each episode, an encoder will first be used to encode the nodes as the node embeddings $e^{(N)}$ Then a decoder will be executed for N times to get the node permutation a. During decoding, the state representation is defined to consist of two parts: all node embeddings $\boldsymbol{e}^{(N)}$, and current context embedding $\boldsymbol{h}_t$, which reflects the state changes. $\boldsymbol{h}_t$ is obtained from $\boldsymbol{e}^{(N)}$, the embedding of the first chosen node $\boldsymbol{e}_1^{(N)}$, and the embedding of the last chosen node $\boldsymbol{e}_{t-1}^{(N)}$:

$$\boldsymbol{h}_t = \text{Concat}\left(\text{Mean}(\boldsymbol{e}^{(N)}); \boldsymbol{e}_1^{(N)}; \boldsymbol{e}_{t-1}^{(N)}\right). \tag{3}$$

Given $\boldsymbol{e}^{(N)}$ and $\boldsymbol{h}_t$, each node candidate is assigned with an attention value for selection. A visitation mask $\boldsymbol{m}_t^{\text{v}}$ is applied to track those visited nodes, and prevent them from being picked again. In other words, $\boldsymbol{m}_t^{\text{v}}$ can be used to specify $a_{1:t-1}$, and we can reformulate $p_\theta(a_t|s_t, a_{1:t-1})$ in Eq. (2) as $p_\theta(a_t|\boldsymbol{h}_t, \boldsymbol{e}^{(N)}, \boldsymbol{m}_t^{\text{v}})$. Fig. 2(a) shows the decoding process of AM. The dashed variables are fixed throughout the episode. The black, blue, red arrows denote the processes of computing context embedding, computing attention value for each action candidate, and updating variables for next time step, respectively.

We then design AM to be suitable for multi-task learning. To yield good generalizability among multiple tasks, it is crucial to learn the knowledge shared by them [58,66,70]. For TSPs, there exists similar dynamics that is invariant to problem sizes. For example, when considering visited nodes and unvisited nodes as sub-graphs, selecting a new node will lead to the one-node-expansion of the former sub-graph, and the corresponding compression of the later one. We treat the similar environment dynamics as shared knowledge, then derive the problem

size-invariant node selection strategy by accurately estimating such dynamics. Since $e^{(N)}$ and $e_1^{(N)}$ in AM are static per episode, and the last chosen node $e_{t-1}^{(N)}$ is the only part that changes over time, which is insufficient to reflect the shared dynamics. We get inspiration from the heterogeneous graph [71,72], where the nodes and / or edges within a graph may have different types. In the case of TSP, the nodes' 2-D coordinates are modeled during input encoding. During solution decoding, the node visitation information can serve as additional node type. The nodes can be divided into sub-graphs based on their different visitation types, then be modeled separately, which better reflects the environment dynamics. Motivated by this, we seek to build dynamics-aware context embedding by taking node visitation information into account. Fig. 2(b) illustrates our modification: when computing $\boldsymbol{h}_t$, we replace $e^{(N)}$ with a new variable, node visitation embedding $e_t^{(V)}$, which represents current node visitation information. After selecting an action a_t, the visitation embedding for next time step $e_{t+1}^{(V)}$ is updated using $e^{(N)}$ and new visitation mask $\boldsymbol{m}_{t+1}^{\mathrm{v}}$.

We design an attentive module to compute $e_t^{(V)}$. We denote the node visitation embedding as the sub-graph representation of either visited nodes or unvisited nodes, or combined representation of both sub-graphs. Take the sub-graph of visited nodes as an example. We reuse the visitation mask $\boldsymbol{m}_t^{\mathrm{v}}$ in action scoring, which is a binary mask with visited nodes masked as 1, and others masked as 0. We first multiply $e^{(\mathrm{N})}$ with $\boldsymbol{m}_t^{\mathrm{v}}$ to obtain the sub-graph representation e_t^{v}:

$$e_t^{\mathrm{v}} = \boldsymbol{m}_t^{\mathrm{v}} \odot e^{(\mathrm{N})} \tag{4}$$

where node embeddings of those unvisited are masked as zero vectors. Then we project $e^{(\mathrm{N})}$ to compute attentive weights, and multiply them with e_t^{v} to compute $e_t^{(V)}$:

$$e_t^{(V)} = \sum \left(\mathrm{Softmax}(W\, e^{(\mathrm{N})}) \odot e_t^{\mathrm{v}}\right) \tag{5}$$

where W is the learnable variable. Different from AM, which obtains graph representation by averaging all node embeddings, we sum the embeddings, since the embeddings have been scaled by attention weights, and the sub-graph may be too sparse at the beginning of the solution decoding. Similarly, $e_t^{(V)}$ can be obtained from the sub-graph of unvisited nodes. In this case, we obtain the graph representation e_t^{unv} by multiplying $(1 - \boldsymbol{m}_t^{\mathrm{v}})$ with $e^{(\mathrm{N})}$. When using both sub-graphs to compute $e_t^{(V)}$, besides computing e_t^{v} and e_t^{unv}, we learn different projection weights to reflect different aggregation preference:

$$\begin{aligned} e_t^{(V)} = &\sum \left(\mathrm{Softmax}(W_v\, e^{(\mathrm{N})}) \odot e_t^{\mathrm{v}}\right) + \\ &\sum \left(\mathrm{Softmax}(W_{unv}\, e^{(\mathrm{N})}) \odot e_t^{\mathrm{unv}}\right) \end{aligned} \tag{6}$$

Finally, the node visitation embedding $e_t^{(V)}$ will be combined with $e_{t-1}^{(N)}$ and $e_1^{(N)}$ to obtain $\boldsymbol{h}_t$.

Given $\boldsymbol{h}_t$ and $e^{(N)}$, the attention value for each node can then be computed via arbitrary attention mechanism. In particular, we follow AM to compute the attention value via a way similar to the multi-head self-attention [73]. We first compute the multi-head attention using $\boldsymbol{h}$ (for simplicity, we omit the subscript t) as the input vector v_Q, and $e^{(N)}$ as both the input vectors v_K and v_V:

$$\boldsymbol{h}^{\mathrm{MHA}} = \mathrm{MHA}(v_Q = \boldsymbol{h}, v_K = e^{(N)}, v_V = e^{(N)}) \tag{7}$$

Then for each node with index $i \in [1, N]$, we compute its attention value $\boldsymbol{u}_i$ based on $\boldsymbol{h}^{\mathrm{MHA}}$ and its node embedding e_i^N,

$$\boldsymbol{u}_i = \begin{cases} C \cdot \tanh\left(\frac{(W_{\mathrm{Q}}\boldsymbol{h}^{\mathrm{MHA}})^T (W_{\mathrm{K}} e_i^N)}{\sqrt{D}}\right) & \text{if } \boldsymbol{m}_t^{\mathrm{v}} = 0 \\ -\infty & \text{otherwise.} \end{cases} \tag{8}$$

where W_{Q} and W_{K} are learnable, and C is a constant for clipping the result. The visitation mask $\boldsymbol{m}_t^{\mathrm{v}}$ is used here to reset the probabilities of those visited as $-\infty$, therefore only those unvisited are regarded as action candidates. The obtained attention values can be treated as probabilities p_θ to sample or greedily select an action. Given the predicted probabilities, some search strategies such as active search and beam search can be applied during inference to increase the capacity of greedy search [17].

4.2. Policy learning

When an episode is finished, the action chosen at each time step $\{a_1, a_2, \ldots, a_t, \ldots, a_N\}$ will be treated as the solution a to the problem instance s, and the episodic reward will be then computed based on Eq. (1). We follow the policy gradient method for model optimization [74]. We formulate the learning objective as the expected cost $\mathcal{L}(s) = \mathbb{E}_{p_\theta(a|s)}[L(a|s)]$, then estimate its gradient to update θ:

$$\nabla_\theta \mathcal{L}(s) = \mathbb{E}_{p_\theta(a|s)}\left[(L(a|s) - b(s))\nabla \log\ p_\theta(a|s)\right] \tag{9}$$

The Monte Carlo sampling over the whole episode may incur high variance, where a baseline estimator $b(s)$ could be adopted for variance reduction. Some baseline estimators, such as the exponential baseline, the critic baseline, and the greedy rollout baseline, have been applied in previous studies [17,21].

4.3. Scheduled data utilization strategy

During training, each epoch contains multiple batch-wise training steps. Joshi et al. [25] has empirically proved that compared to training with fixed problem size (e.g., TSP50), training with a wide range of smaller problem sizes (e.g., TSP20-50) helps to retain performance, while enjoying faster running speed. For each training step, a problem size is sampled randomly to build a batch of TSP instances. However, this training strategy ignores the task similarity. Although TSP tasks have same goal and similar dynamics, different problem sizes result in different similarities of task pairs. From the perspective of multi-task learning [28,29], highly similar tasks should be learnt together, but not those largely different from each other. We borrow this idea and extend it to TSP with different problem sizes: tasks with similar problem sizes are learnt in closer training steps. In each training epoch, we schedule the tasks to be with an ascending problem size order. For an epoch with E steps, and T tasks (i.e., T problem sizes), the model will be trained on a task for continuously $\lfloor \frac{E}{T} \rfloor$ steps, then be switched to another task by the order of problem sizes. The benefit of this data utilization strategy is that it makes tasks trained on closed steps be with high similarity, thus stabilize learning. In terms of curriculum learning, we regard larger TSPs to be more difficult to learn, therefore forming a curricula schedule with gradually increasing difficulties would be helpful for learning.

5. Experiments

5.1. Experiment setting

We carry out the experiments on three branches: (1) Training and evaluating on the fixed problem size. (2) Zero-shot generalization to unseen larger problem sizes. (3) Few-shot generalization with a few fine-tuning samples of the target problem size. For (1), we follow [20,21] to train the model on TSP20, TSP50 and TSP100, respectively, and then evaluate the model on the same problem size. For (2) and (3), we follow Joshi et al. [25] to train the model on mixed problem sizes ranging from 20 to 50 (we denote this setting as "TSP20-50"), and then evaluate the model on TSP20, TSP50 and larger TSPs.

As no RL-based model is specifically designed for multi-task setting, we use AM [21], one of the state-of-the-art and popular models under the setting of fixed problem size, as our major baseline. We design three variants with different kinds of visitation information:

- **AM-V**, which constructs $e_t^{(V)}$ with sub-graph of visited nodes.
- **AM-UNV**, which constructs $e_t^{(V)}$ with sub-graph of unvisited nodes.
- **AM-VUNV**, which constructs $e_t^{(V)}$ with both sub-graphs.

Table 1
The evaluating performance (optimality gap) of models after training. "p-value" measures the statistical significance at the 95% confidence interval between the baseline AM and our models.

Method	TSP20		TSP50		TSP100	
	Opt.gap (%)	p-value	Opt.gap (%)	p-value	Opt.gap (%)	p-value
AM	2.24 ± 0.25	–	5.10 ± 0.17	–	8.03 ± 0.30	–
AM-V	1.88 ± 0.12	0.0000	4.57 ± 0.08	0.0000	7.93 ± 0.16	0.0177
AM-UNV	1.88 ± 0.12	0.0000	**4.43**±0.08	0.0000	7.75 ± 0.21	0.0000
AM-VUNV	**1.69**±0.12	0.0000	4.63 ± 0.08	0.0000	**7.52**±0.11	0.0000

5.2. Implementation and training details

We implement our models upon AM's official implementation with similar hyper-parameter settings to yield a fair comparison.[1] We set the encoder to be with 3 transformer blocks. For each block, we set the multi-head attention sub-module with 8 heads, and set the feed-forward networks sub-module with 512 hidden units. We set the dimension of the node embedding as 128. Regarding the decoder, the attention values for action selection are computed through using an 8-head attention sub-module and a single-head attention sub-module. Instead of using greedy rollout baseline or critic baseline, we apply a simple non-learnable exponential baseline estimator, which computes the moving average exponentiality $b(s) = M$ with decay β. Specifically, we define an iteration as a training step within an epoch, then define M as the loss $L(a|s)$ of the first iteration. In the successive iterations, we update M through $M \leftarrow \beta M + (1-\beta)L(a|s)$. This baseline speeds up training and helps us to isolate the influence of baseline estimating.

For all problem instances, we generate the node coordinates uniformly within $[0, 1]^2$. During training, we follow similar hyper-parameter setting in [25]. An training epoch consists of 128,000 problem instances, which are generated on the fly. The models will be trained for 100 epochs, resulting in a total of 12,800,000 training problem instances. We set batch size as 128, which means that for each task, there are 32 training steps per epoch. The models will be optimized using the Adam optimizer. The learning rate is set as 0.0001, where no decay is applied.

We consider 10 fixed evaluation sets: TSP20, TSP50, TSP70, TSP100, TSP120, TSP150, TSP170, TSP200, TSP220 and TSP250, each with 1280 instances. After every training epoch, we evaluate the models on those evaluation sets, with the batch size being set as 32. During evaluating, we let the models select the nodes greedily — the node candidate with the highest attention value will be chosen. For simplicity, we do not apply search heuristics such as beam search and 2-opt.

During fine-tuning, we load the pre-trained model and re-train it on the target task with fixed problem size for at most 10 epochs. The batch size is set as 128. The learning rate is set as 0.00001, where no decay is applied.

5.3. Evaluation metrics

We use the optimality gap to measure the performance of models. Specifically, we normalize the predicted tour length with the optimal tour length for this problem instance:

$$\text{Gap}_L = \frac{L - L_{\text{opt}}}{L_{\text{opt}}} \times 100\% \tag{10}$$

Smaller optimality gap means better performance. We obtain the mean optimal tour length for each evaluation set using the exact solver Concorde [75]. We run each experiment with three random seeds and report the average result.

6. Results and discussions

6.1. Fixed problem size

Before evaluating the generalizability on larger TSPs, we start with training and evaluating models on same and fixed problem size. Fig. 3 shows the performance with respect to training epochs on TSP20, TSP50 and TSP100, and Table 1 compares the final performance after training. We use the paired sample t-test to measure the statistical significance at the 95% confidence interval between the baseline AM and our models — the performance of our model could be treated as significantly different from AM if the p-value is less than 0.05. We regard an RL model as more sample-efficient, if this model costs fewer interaction data[2] to achieve comparable performance to the baseline models. The proposed models (AM-V, AM-UNV and AM-VUNV), by taking into account visitation information, achieve higher sample efficiency than AM. Their final performance after 100 epochs are better than AM as well. The performance difference between the three models and AM is more significant on larger problems, while the difference between the three models themselves (i.e., between the different types of visitation information) is marginal.

6.2. Zero-shot generalization

In this section, we evaluate zero-shot generalization on different problem sizes. We train all models on TSP20-50 for 100 epochs. We use two data utilization strategies: (1) SampledData, where each training batch is generated with a randomly sampled problem size. (2) OrderedData, where training batches within each epoch are scheduled in ascending order of the problem sizes. Fig. 4 and Table 2 compares the performance with respect to problem sizes under the two strategies. Under both strategies, the model considering the sub-graph of unvisited nodes (AM-UNV) performs better than AM. The other two models with consideration of visited sub-graph, AM-V and AM-VUNV, show similar or a little worse performance than AM when using "SampledData". However, with "OrderedData" strategy, their performance become the best among all models, especially in larger problems such as TSP250.

We further investigate the influence of data utilization strategy. To this end, we compare different training methods for each model, with results shown in Fig. 5. Similar to Joshi et al. [25], models trained on TSP20-50 partly retain the performance of those trained on TSP50, while enjoying faster training speed due to smaller problem sizes. Compared to randomly sampled training problem sizes (e.g., pink lines), scheduling problem sizes in order (e.g., blue lines) helps to make the performance of TSP20-50 closer to TSP50 (e.g., gold lines). The proposed models with visited node sub-graph (AM-V, AM-VUNV) enjoy most performance gain. A possible reason is that it would be easier for the model to capture shared dynamics (e.g., visited nodes) if closely-trained tasks are similar. Based on Figs. 6 and 7, we conclude that considering visitation information helps to improve zero-shot generalizability to unseen larger TSPs. However, the data utilization strategy matters when the model is trained under multi-task learning setting — scheduling training tasks by their problem sizes leads to large performance improvement. We note that the performance in our experiments

[1] https://github.com/wouterkool/attention-learn-to-route

[2] In our experiment setting, this is equivalent to fewer epochs.

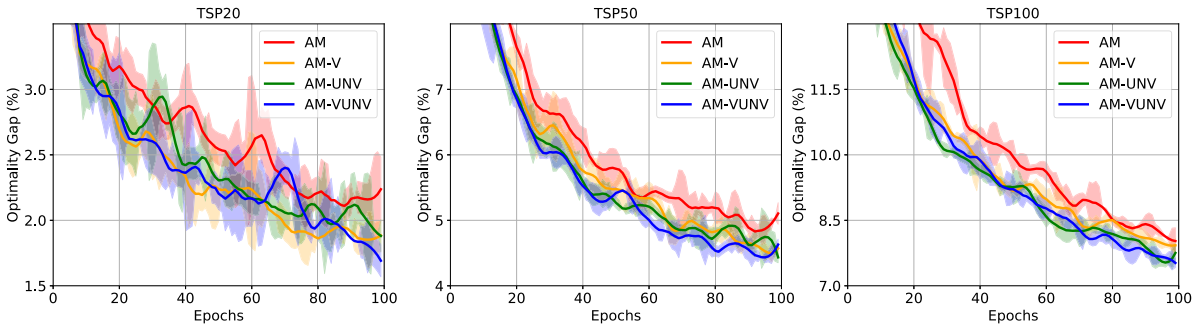


Fig. 3. The evaluating performance of models with respect to training epochs. The training and evaluating datasets share the same problem size. The optimality gaps of models with visitation information (AM-V, AM-UNV and AM-VUNV) drop faster than AM. The models also achieve better final performance than AM. The shadowed area denotes standard deviation.

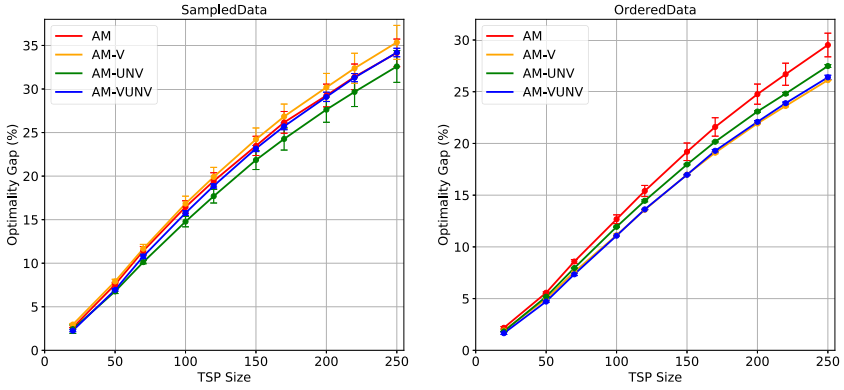


Fig. 4. The zero-shot generalization performance with respect to problem sizes. All models are trained on TSP20-50 for 100 epochs. "SampledData" and "OrderedData" (proposed) denote two training data utilization strategies. The dot denotes particular evaluation dataset (e.g., TSP220), and the error bar denotes standard deviation. Under both strategies, the model with sub-graph of unvisited nodes (AM-UNV) consistently shows better performance than AM. Under the "OrderedData" strategy, all models with node visitation information significantly outperform AM.

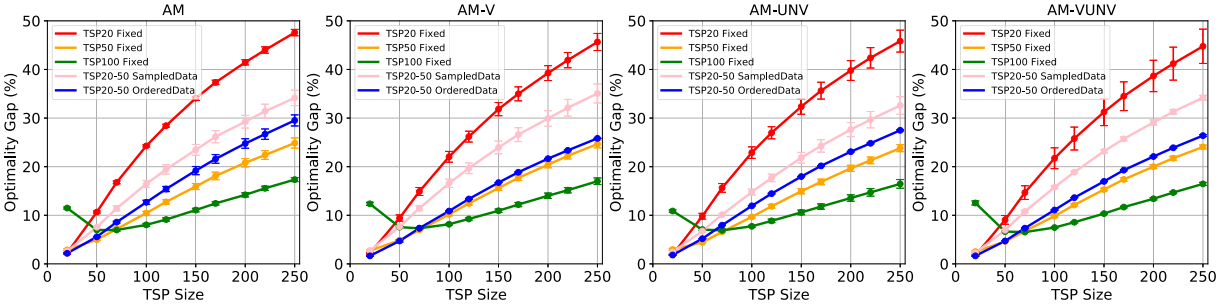


Fig. 5. The zero-shot generalization performance of different training methods. The data utilization strategy "OrderedData" helps to improve the performance in multi-task training (TSP20-50). The performance gain is most significant in models considering sub-graph of visited nodes (AM-V and AM-VUNV).

is generally worse than those reported in Joshi et al. [25]. This is due to two reasons: (1) Their work applied graph sparsification heuristics to construct sub-graph based on each node's nearest neighbors. (2) the exponential baseline we used, which speeds up training but leads to worse performance than the greedy rollout baseline [21]. We leave the comparison with graph sparsification heuristics as well as other policy gradient baselines as future work.

6.3. Few-shot generalization

We then investigate whether our models can achieve better performance improvement given a few fine-tuning samples of target problem size. We load the models pre-trained on TSP20-50 for 100 epochs with "OrderedData" strategy, then re-train them on target problem size for up to 10 epochs. Based on the encoder–decoder framework, we consider following four fine-tuning types:

- **ftEncftDec**, which loads the pre-trained encoder and decoder, then fine-tunes both of them.
- **fixEncftDec**, which loads the pre-trained encoder and decoder, then fine-tunes the decoder only, with the encoder being fixed.
- **ftEnctrDec**, which loads and fine-tunes the pre-trained encoder, and trains the decoder from scratch.
- **fixEnctrDec**, which loads the pre-trained encoder and keeps it fixed, then trains the decoder from scratch.

Fig. 6 and Table 3 show the performance with respect to problem sizes. Generally, fine-tuning the pre-trained encoder shows better performance than loading it without fine-tuning ("ftEncftDec" v.s. "fixEncftDec", "ftEnctrDec" v.s. "fixEnctrDec"). Regarding the decoder,

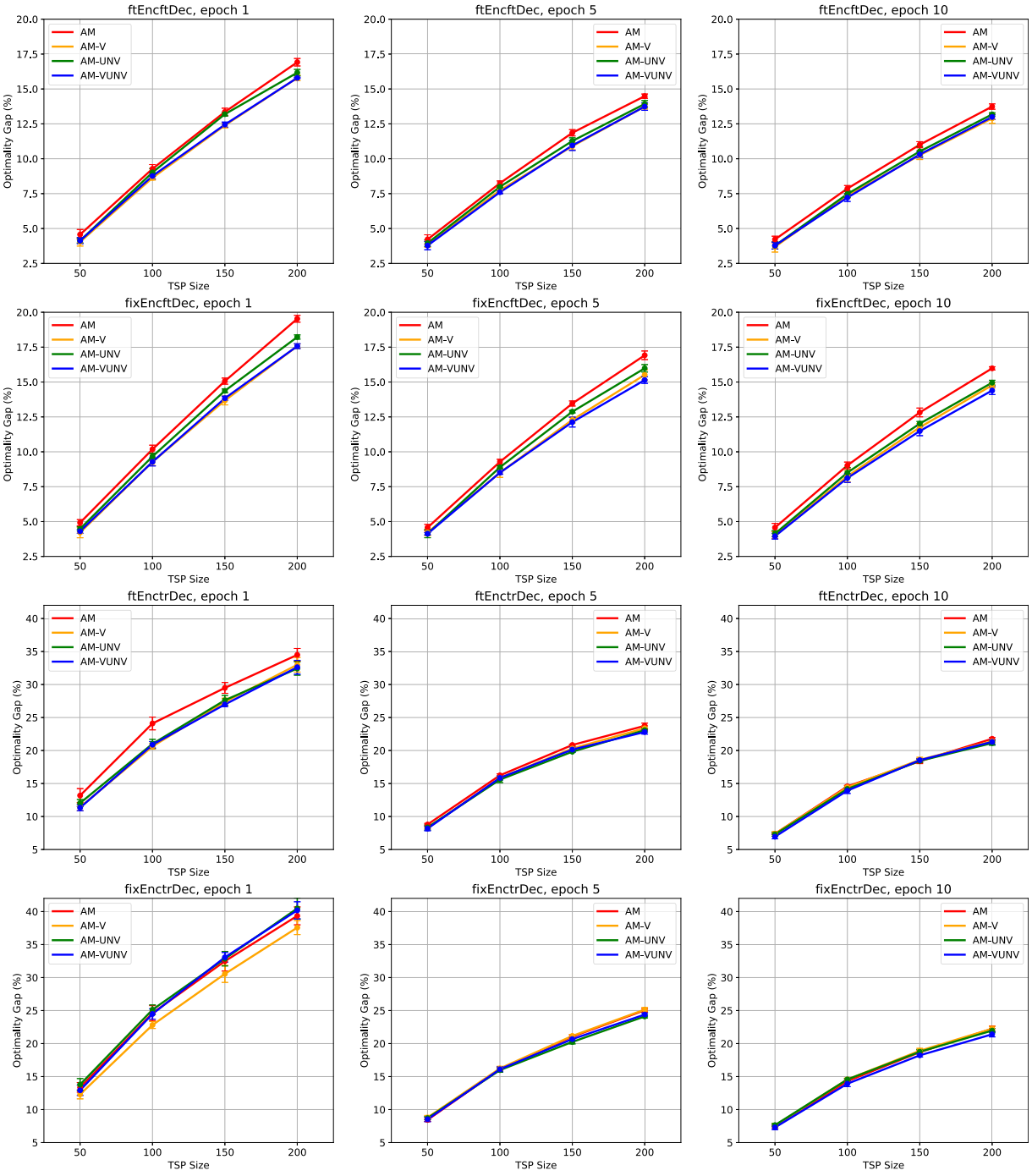


Fig. 6. The few-shot generalization performance of models fine-tuned on target problem sizes (TSP50, TSP100, TSP150, TSP200) for 1 epoch, 5 epochs and 10 epochs. "ftEncftDec", "fixEncftDec", "ftEnctrDec" and "fixEnctrDec" denote different fine-tuning types, where "ftEncftDec" is with the best performance. Two of the proposed models, AM-V and AM-VUNV, outperform AM throughout the fine-tuning process.

fine-tuning the pre-trained decoder leads to much better performance than training a decoder from scratch ("ftEncftDec" v.s. "ftEnctrDec", "fixEncftDec" v.s. "fixEnctrDec"). In experiments with the two better-performed fine-tuning types ("ftEncftDec", "fixEncftDec"), two of the proposed models, AM-V and AM-VUNV, show better adaptation performance than AM from the beginning of fine-tuning process (1 epoch). When being fine-tuned for 10 epochs, all three proposed models consistently outperform AM in all four target problem sizes. Fig. 7 shows the performance on TSP200 with respect to epochs. Similar to previous observation, the optimality gap of proposed models is lower than AM at the beginning (i.e., zero-shot), and decreases lower with the progress of fine-tuning. In terms of different types of node visitation information, the sub-graph of visited nodes seems to be more important than unvisited nodes. As "ftEncftDec" is with the best performance, we use this fine-tuning type in the later sections.

TSPs in real world [76] may contain more than thousands of nodes, which require huge GPU memory to run with large batch size. It is worthwhile to investigate whether the proposed models work well with small fine-tuning batch size. In order to achieve this, we fine-tune these models on TSP200 with different batch sizes. Fig. 8 shows the result with the batch size being reduced from 128 to 64 and to 32. It can be observed that the proposed models, especially AM-V and AM-VUNV, consistently outperform AM with the fine-tuning progress. Regarding the stability, the proposed models are robust with respect to batch size,

Table 2
The zero-shot generalization performance.

Method	TSP50		TSP100		TSP150		TSP200	
	SampledData							
	Opt.gap (%)	p-value	Opt.gap (%)	p-value	Opt.gap (%)	p-value	Opt.gap (%)	p-value
AM	7.56 ± 0.25	–	16.45 ± 0.72	–	23.47 ± 1.10	–	29.25 ± 1.30	–
AM-V	7.91 ± 0.25	0.0000	16.84 ± 0.86	0.0022	24.22 ± 1.31	0.0001	30.19 ± 1.60	0.0001
AM-UNV	**6.77**±0.25	0.0000	**14.78**±0.60	0.0000	**21.85**±1.09	0.0000	**27.63**±1.45	0.0000
AM-VUNV	6.94 ± 0.14	0.0000	15.76 ± 0.26	0.0000	23.17 ± 0.35	0.0497	29.10 ± 0.54	0.5631
	OrderedData							
Method	TSP50		TSP100		TSP150		TSP200	
	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value
AM	5.57 ± 0.08	–	12.67 ± 0.43	–	19.20 ± 0.85	–	24.77 ± 0.97	–
AM-V	4.98 ± 0.22	0.0000	11.13 ± 0.12	0.0000	16.99 ± 0.00	0.0000	**21.93**±0.04	0.0000
AM-UNV	5.18 ± 0.00	0.0000	11.94 ± 0.06	0.0000	17.97 ± 0.05	0.0000	23.08 ± 0.08	0.0000
AM-VUNV	**4.72**±0.08	0.0000	**11.08**±0.00	0.0000	**16.97**±0.05	0.0000	22.09 ± 0.12	0.0000

Table 3
The few-shot generalization performance, all models are fine-tuned on target problem sizes for 10 epochs.

Method	TSP50		TSP100		TSP150		TSP200	
	ftEncftDec							
	Opt.gap (%)	p-value	Opt.gap (%)	p-value	Opt.gap (%)	p-value	Opt.gap (%)	p-value
AM	4.22 ± 0.23	–	7.86 ± 0.22	–	11.00 ± 0.22	–	13.74 ± 0.19	–
AM-V	**3.69**±0.38	0.0000	7.35 ± 0.16	0.0000	**10.26**±0.31	0.0000	**12.90**±0.36	0.0000
AM-UNV	3.78 ± 0.25	0.0000	7.47 ± 0.24	0.0000	10.52 ± 0.35	0.0000	13.18 ± 0.12	0.0000
AM-VUNV	3.78 ± 0.25	0.0000	**7.22**±0.28	0.0000	10.31 ± 0.22	0.0000	12.99 ± 0.18	0.0000
	fixEncftDec							
Method	TSP50		TSP100		TSP150		TSP200	
	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value
AM	4.57 ± 0.30	–	9.02 ± 0.24	–	12.82 ± 0.31	–	15.98 ± 0.12	–
AM-V	4.04 ± 0.25	0.0000	8.25 ± 0.40	0.0000	11.75 ± 0.27	0.0000	14.77 ± 0.19	0.0000
AM-UNV	4.13 ± 0.22	0.0000	8.51 ± 0.37	0.0000	12.02 ± 0.15	0.0000	14.95 ± 0.18	0.0000
AM-VUNV	**3.95**±0.21	0.0000	**8.12**±0.32	0.0000	**11.49**±0.33	0.0000	**14.39**±0.29	0.0000
	ftEnctrDec							
Method	TSP50		TSP100		TSP150		TSP200	
	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value
AM	7.38 ± 0.08	–	14.56 ± 0.16	–	**18.38**±0.36	–	21.78 ± 0.22	–
AM-V	7.38 ± 0.32	0.6022	14.43 ± 0.22	0.0000	18.59 ± 0.38	0.0003	21.40 ± 0.20	0.0000
AM-UNV	**7.29**±0.29	0.0013	14.18 ± 0.21	0.0000	18.43 ± 0.28	0.2621	**21.12**±0.32	0.0000
AM-VUNV	6.94 ± 0.33	0.0000	**13.92**±0.43	0.0000	18.54 ± 0.18	0.0001	21.31 ± 0.38	0.0000
	fixEnctrDec							
Method	TSP50		TSP100		TSP150		TSP200	
	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value	Opt.Gap (%)	p-value
AM	7.38 ± 0.23	–	14.30 ± 0.23	–	18.70 ± 0.23	–	22.24 ± 0.40	–
AM-V	7.56 ± 0.36	0.0003	14.56 ± 0.21	0.0000	18.91 ± 0.36	0.0000	22.24 ± 0.36	0.9573
AM-UNV	7.64 ± 0.23	0.0000	14.56 ± 0.16	0.0000	18.75 ± 0.35	0.3804	21.96 ± 0.33	0.0000
AM-VUNV	**7.29**±0.33	0.0272	**13.92**±0.45	0.0000	**18.22**±0.24	0.0000	**21.40**±0.40	0.0000

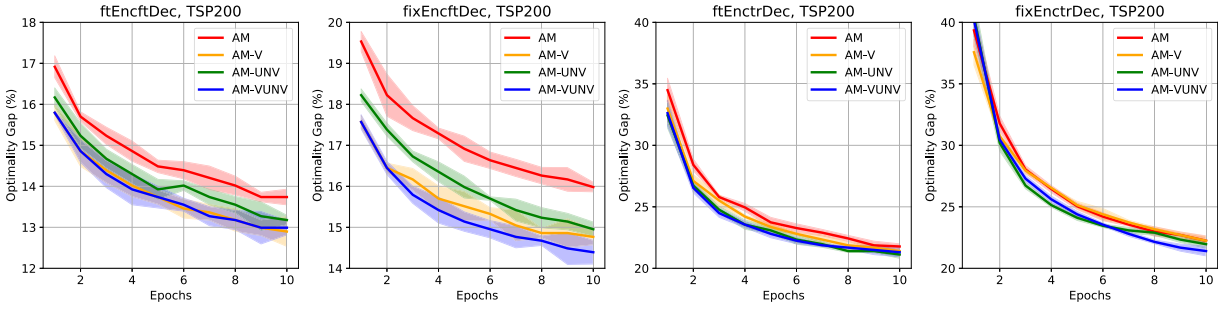


Fig. 7. The few-shot generalization performance of models with respect to fine-tuning epochs on TSP200.

in the sense that they have similar performance standard deviations. In addition, we find that models with smaller fine-tuning batch size show better performance. This may be a result of more updating steps per epoch, which, however, also leads to longer fine-tuning time.

Finally, we compare the contribution of encoder and decoder. While the above experiments have demonstrated the effectiveness of learning shared dynamics during decoding, we are interested in finding out whether modeling such dynamics also improves the encoder to help

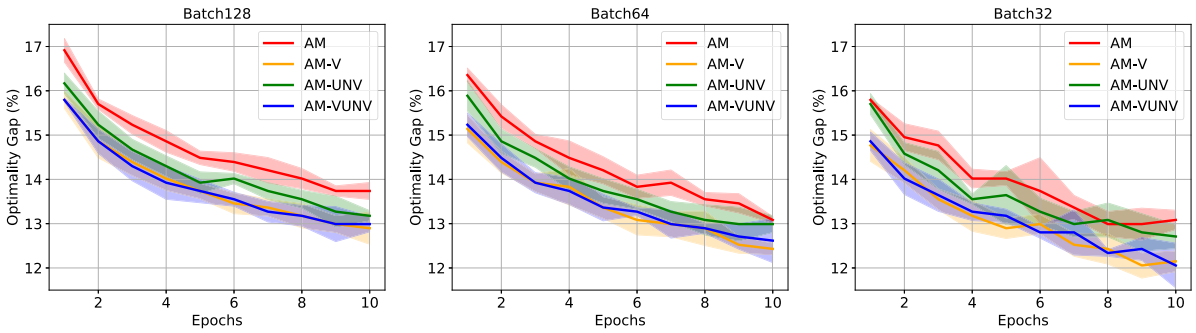


Fig. 8. The few-shot generalization performance of models with different fine-tuning batch sizes on TSP200. Under smaller batch sizes, the proposed models show consistently better performance than AM.

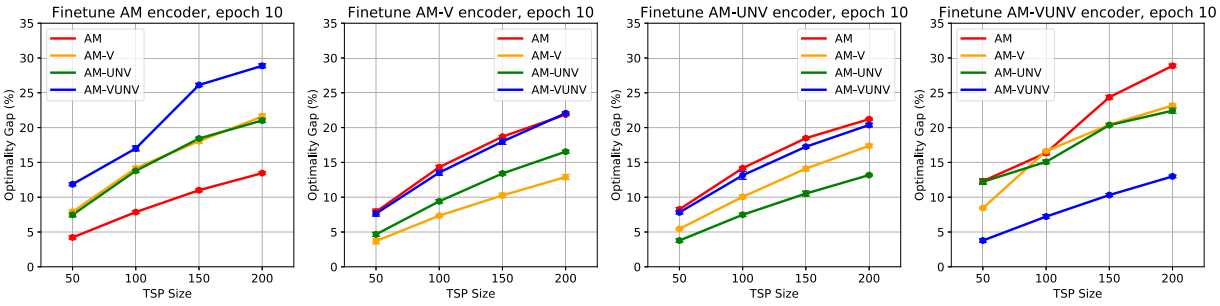


Fig. 9. The few-shot generalization performance of models fine-tuned on target problem sizes (TSP50, TSP100, TSP150, TSP200) for 10 epochs. The models are with same pre-trained encoder (e.g., AM encoder) and their respective decoders.

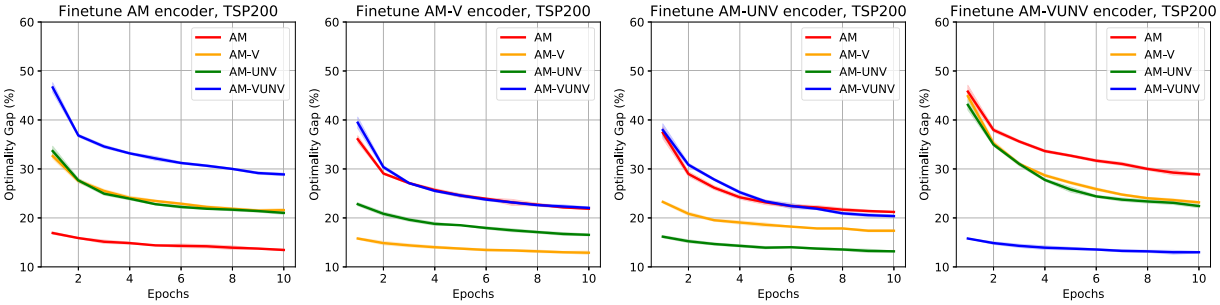


Fig. 10. The few-shot generalization performance of models with respect to fine-tuning epochs on TSP200. The models are with same pre-trained encoder (e.g., AM encoder) and their respective decoders.

achieve performance gain. For this purpose, we load the same pre-trained encoder (e.g., AM's encoder) for all models, and their respective decoders, then fine-tune both the encoder and decoder (i.e., "ftEncft-Dec"). Fig. 9 shows the result on different target problem sizes, and Fig. 10 shows the result on TSP200 with respect to fine-tuning epochs. For proposed models, loading AM's pre-trained encoder leads to significantly performance drop, making their optimality gap much higher than AM. This observation indicates that the performance gain of the proposed models is not only attributed to capturing the environment dynamics during decoding, but also a result of the improved encoder. Considering environment dynamics helps the encoder to obtain better node embeddings and construct a better "global view" of the problem, contributing to performance gain in return. Loading pre-trained encoder of AM-V or AM-UNV shows relatively better performance. Besides, comparing to AM and AM-VUNV, AM-V and AM-UNV retain the performance most when they load each other's pre-trained encoder. A possible reason is that these two models are with similar decoder architecture.

7. Conclusion

In this work, we studied generalizing RL-based TSP solver to unseen larger problems. We formulated the problem as multi-task learning, and identified the objective as learning dynamics shared by TSP tasks of different problem sizes. Attention-based models are designed to capture node visitation information, which could accurately model such dynamics. A scheduled data utilization strategy is also proposed to stabilize learning in multi-task setting. Through extensive experiments, our model is proved to obtain improved generalizability in both zero-shot and few-shot learning cases.

It should be noticed that under different experimental conditions, the proposed models with different types of visitation information perform differently. In experiments where all training instances and evaluating instances are with same problem size, the model with consideration to both visited and unvisited sub-graphs (AM-VUNV) has the lowest optimality gap. However, in zero-shot and few-shot experiments (with data utilization strategy "OrderData"), only considering the sub-graph of visited nodes (AM-V) has similar optimality gap with

AM-VUNV, but with higher generalizability than AM-UNV. A possible reason is that TSP instances with different problem sizes share similar dynamics in the sub-graph of visited nodes, especially at the beginning of route generation.

In future, this work can be further extended from two dimensions. On the one hand, to facilitate its adoption in real-life scenarios, more complex constraints (e.g., capacities and time windows) can be considered in the model. On the other hand, this model can be further extended to many other combinatorial optimization problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by National Natural Science Foundation of China (72174042) and the Natural Science Foundation of Guangdong Province (2023A1515011402).

References

[1] H. Qin, X. Su, T. Ren, Z. Luo, A review on the electric vehicle routing problems: Variants and algorithms, Front. Eng. Manag. 8 (2021) 370–389.
[2] I.I. Huerta, D.A. Neira, D.A. Ortega, V. Varas, J. Godoy, R. Asín-Achá, Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection, Expert Syst. Appl. 187 (2022) 115948.
[3] M.M. Flood, The traveling-salesman problem, Oper. Res. 4 (1) (1956) 61–75.
[4] G. Dantzig, R. Fulkerson, S. Johnson, Solution of a large-scale traveling-salesman problem, J. Oper. Res. Soc. Am. 2 (4) (1954) 393–410.
[5] G. Laporte, The traveling salesman problem: An overview of exact and approximate algorithms, European J. Oper. Res. 59 (2) (1992) 231–247.
[6] R. Roberti, M. Ruthmair, Exact methods for the traveling salesman problem with drone, Transp. Sci. 55 (2) (2021) 315–335.
[7] R. Durbin, D. Willshaw, An analogue approach to the travelling salesman problem using an elastic net method, Nature 326 (6114) (1987) 689–691.
[8] G. Erdoğan, E.A. Yıldırım, Exact and heuristic algorithms for the Carrier–Vehicle traveling salesman problem, Transp. Sci. 55 (1) (2021) 101–121.
[9] Y. Zhou, W. Xu, Z.H. Fu, M. Zhou, Multi-neighborhood simulated annealing-based iterated local search for colored traveling salesman problems, IEEE Trans. Intell. Transp. Syst. (2022).
[10] M. Das, A. Roy, S. Maity, S. Kar, A quantum-inspired ant colony optimization for solving a sustainable four-dimensional traveling salesman problem under type-2 fuzzy variable, Adv. Eng. Inform. 55 (2023) 101816.
[11] K. Leksakul, U. Smutkupt, R. Jintawiwat, S. Phongmoo, Heuristic approach for solving employee bus routes in a large-scale industrial factory, Adv. Eng. Inform. 32 (2017) 176–187.
[12] C.H. Chen, Y.C. Lee, A.Y. Chen, A building information model enabled multiple traveling salesman problem for building interior patrols, Adv. Eng. Inform. 47 (2021) 101237.
[13] S. Shao, G. Xu, M. Li, G.Q. Huang, Synchronizing e-commerce city logistics with sliding time windows, Transp. Res. Part E: Logist. Transp. Rev. 123 (2019) 17–28.
[14] S.H. Huang, Y.H. Huang, C.A. Blazquez, C.Y. Chen, Solving the vehicle routing problem with drone for delivery services using an ant colony optimization algorithm, Adv. Eng. Inform. 51 (2022) 101536.
[15] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, European J. Oper. Res. 290 (2) (2021) 405–421.
[16] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2692–2700.
[17] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, 2016, arXiv preprint arXiv:1611.09940.
[18] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, C. Zhang, Reinforcement learning with multiple relational attention for solving vehicle routing problems, IEEE Trans. Cybern. 52 (10) (2021) 11107–11120.
[19] A. Zhu, T. Dai, G. Xu, P. Pauwels, B. de Vries, M. Fang, Deep reinforcement learning for real-time assembly planning in robot-based prefabricated construction, IEEE Trans. Autom. Sci. Eng. (2023).
[20] M. Nazari, A. Oroojlooy, L. Snyder, M. Takác, Reinforcement learning for solving the vehicle routing problem, in: Advances in Neural Information Processing Systems, 2018, pp. 9839–9849.
[21] W. Kool, H. van Hoof, M. Welling, Attention, learn to solve routing problems! in: International Conference on Learning Representations, 2019, URL https://openreview.net/forum?id=ByxBFsRqYm.
[22] Z. Zhang, H. Liu, M. Zhou, J. Wang, Solving dynamic traveling salesman problems with deep reinforcement learning, IEEE Trans. Neural Netw. Learn. Syst. (2021).
[23] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, L.M. Rousseau, Learning heuristics for the tsp by policy gradient, in: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2018, pp. 170–181.
[24] L. Xin, W. Song, Z. Cao, J. Zhang, NeuroLKH: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem, Adv. Neural Inf. Process. Syst. 34 (2021).
[25] C.K. Joshi, Q. Cappart, L.M. Rousseau, T. Laurent, X. Bresson, Learning TSP requires rethinking generalization, 2020, arXiv preprint arXiv:2006.07054.
[26] R. Caruana, Multitask learning, Mach. Learn. 28 (1) (1997) 41–75.
[27] Y. Zhang, Q. Yang, An overview of multi-task learning, Natl. Sci. Rev. 5 (1) (2018) 30–43.
[28] L.T. Liu, U. Dogan, K. Hofmann, Decoding multitask DQN in the world of minecraft, in: The 13th European Workshop on Reinforcement Learning, EWRL 2016, 2016, URL https://www.microsoft.com/en-us/research/publication/decoding-multitask-dqn-in-the-world-of-minecraft/.
[29] T. Standley, A.R. Zamir, D. Chen, L. Guibas, J. Malik, S. Savarese, Which tasks should be learned together in multi-task learning? 2019, arXiv preprint arXiv:1905.07553.
[30] J.F. Cordeau, G. Ghiani, E. Guerriero, Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem, Transp. Sci. 48 (1) (2014) 46–58.
[31] M. Boccia, A. Masone, A. Sforza, C. Sterle, A column-and-row generation approach for the flying sidekick travelling salesman problem, Transp. Res. C 124 (2021) 102913.
[32] X. Xu, J. Li, M. Zhou, Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems, IEEE Trans. Intell. Transp. Syst. 22 (3) (2020) 1583–1593.
[33] F. Carrabs, C. Cerrone, R. Cerulli, B. Golden, An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem, INFORMS J. Comput. 32 (4) (2020) 1030–1048.
[34] S.T.W. Mara, A.P. Rifai, B.M. Sopha, An adaptive large neighborhood search heuristic for the flying sidekick traveling salesman problem with multiple drops, Expert Syst. Appl. (2022) 117647.
[35] J.J. Hopfield, D.W. Tank, "Neural" computation of decisions in optimization problems, Biol. Cybernet. 52 (3) (1985) 141–152.
[36] L.M. Gambardella, M. Dorigo, Ant-Q: A reinforcement learning approach to the traveling salesman problem, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 252–260.
[37] J.P.Q. dos Santos, F.C. de Lima, R.M. Magalhaes, J.D. de Melo, A.D.D. Neto, A parallel hybrid implementation using genetic algorithm, GRASP and reinforcement learning, in: 2009 International Joint Conference on Neural Networks, IEEE, 2009, pp. 2798–2803.
[38] F. Liu, G. Zeng, Study of genetic algorithm with reinforcement learning to solve the TSP, Expert Syst. Appl. 36 (3) (2009) 6995–7001.
[39] J.E. Bell, P.R. McMullen, Ant colony optimization techniques for the vehicle routing problem, Adv. Eng. Inform. 18 (1) (2004) 41–48.
[40] J. He, W. Zhang, Y. Huang, W. Yan, A simulation optimization method for internal trucks sharing assignment among multiple container terminals, Adv. Eng. Inform. 27 (4) (2013) 598–614.
[41] Y. Kaempfer, L. Wolf, Learning the multiple traveling salesmen problem with permutation invariant pooling networks, 2018, arXiv preprint arXiv:1803.09621.
[42] J. Devlin, M.W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186.
[43] M. Prates, P.H. Avelar, H. Lemos, L.C. Lamb, M.Y. Vardi, Learning to solve np-complete problems: A graph neural network for decision tsp, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, (no. 01) 2019, pp. 4731–4738.
[44] Y. Hu, Z. Zhang, Y. Yao, X. Huyan, X. Zhou, W.S. Lee, A bidirectional graph neural network for traveling salesman problems on arbitrary symmetric graphs, Eng. Appl. Artif. Intell. 97 (2021) 104061.
[45] B. Hudson, Q. Li, M. Malencia, A. Prorok, Graph neural network guided local search for the traveling salesperson problem, 2021, arXiv preprint arXiv:2110.05291.
[46] H. Dai, E.B. Khalil, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 6351–6361.

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

[48] I. Drori, A. Kharkar, W.R. Sickinger, B. Kates, Q. Ma, S. Ge, E. Dolev, B. Dietrich, D.P. Williamson, M. Udell, Learning to solve combinatorial optimization problems on real-world graphs in linear time, 2020, arXiv preprint arXiv:2006.03750.

[49] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020).

[50] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, C. Zhang, Reinforcement learning with multiple relational attention for solving vehicle routing problems, IEEE Trans. Cybern. (2021) 1–14, http://dx.doi.org/10.1109/TCYB.2021.3089179.

[51] X. Chen, Y. Tian, Learning to perform local rewriting for combinatorial optimization, in: Advances in Neural Information Processing Systems, 2019, pp. 6278–6289.

[52] P.R.d.O. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning, 2020, arXiv preprint arXiv:2004.01608.

[53] Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning improvement heuristics for solving routing problems, IEEE Trans. Neural Netw. Learn. Syst. (2021).

[54] Q. Cappart, T. Moisan, L.M. Rousseau, I. Prémont-Schwarz, A. Cire, Combining reinforcement learning and constraint programming for combinatorial optimization, 2020, arXiv preprint arXiv:2006.01610.

[55] K. Li, T. Zhang, R. Wang, Deep reinforcement learning for multiobjective optimization, IEEE Trans. Cybern. (2020).

[56] J. Pasha, A.L. Nwodu, A.M. Fathollahi-Fard, G. Tian, Z. Li, H. Wang, M.A. Dulebenets, Exact and metaheuristic algorithms for the vehicle routing problem with a factory-in-a-box in multi-objective settings, Adv. Eng. Inform. 52 (2022) 101623.

[57] B. Peng, J. Wang, Z. Zhang, A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems, 2020, arXiv preprint arXiv:2002.03282.

[58] A. Maurer, M. Pontil, B. Romera-Paredes, The benefit of multitask representation learning, J. Mach. Learn. Res. 17 (1) (2016) 2853–2884.

[59] E. Parisotto, J.L. Ba, R. Salakhutdinov, Actor-mimic: Deep multitask and transfer reinforcement learning, 2015, arXiv preprint arXiv:1511.06342.

[60] A.A. Rusu, S. Gomez Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, R. Hadsell, Policy distillation, 2015, arXiv. arXiv–1511.

[61] W.M. Czarnecki, R. Pascanu, S. Osindero, S. Jayakumar, G. Swirszcz, M. Jaderberg, Distilling policy distillation, in: The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS, PMLR, 2019, pp. 1331–1340.

[62] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, G. Cai, D. Filliat, N. Díaz-Rodríguez, DISCORL: Continual reinforcement learning via policy distillation, in: NeurIPS Workshop on Deep Reinforcement Learning, 2019.

[63] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, H. van Hasselt, Multi-task deep reinforcement learning with popart, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 3796–3803.

[64] A. Gupta, C. Devin, Y. Liu, P. Abbeel, S. Levine, Learning invariant feature spaces to transfer skills with reinforcement learning, 2017, arXiv preprint arXiv:1703.02949.

[65] N. Makondo, B. Rosman, O. Hasegawa, Accelerating model learning with inter-robot knowledge transfer, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, 2018, pp. 2417–2424, http://dx.doi.org/10.1109/ICRA.2018.8461218.

[66] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, J. Peters, Sharing knowledge in multi-task deep reinforcement learning, in: International Conference on Learning Representations, 2019.

[67] C.F. Perez, F.P. Such, T. Karaletsos, Generalized hidden parameter MDPs transferable model-based RL in a handful of trials, 2020, arXiv preprint arXiv:2002.03072.

[68] A. Zhang, S. Sodhani, K. Khetarpal, J. Pineau, Multi-task reinforcement learning as a hidden-parameter block MDP, 2020, arXiv preprint arXiv:2007.07206.

[69] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018, URL https://openreview.net/forum?id=rJXMpikCZ.

[70] Y. Zhang, Q. Yang, A survey on multi-task learning, 2017, arXiv preprint arXiv:1707.08114.

[71] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, P.S. Yu, Heterogeneous graph attention network, in: The World Wide Web Conference, 2019, pp. 2022–2032.

[72] C. Zhang, D. Song, C. Huang, A. Swami, N.V. Chawla, Heterogeneous graph neural network, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 793–803.

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[74] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (3–4) (1992) 229–256.

[75] D. Applegate, R. Bixby, V. Chvatal, W. Cook, Concorde tsp solver, 2006, 2006, URL http://www.tsp.gatech.edu/concorde.

[76] G. Reinelt, TSPLIB—A traveling salesman problem library, ORSA J. Comput. 3 (4) (1991) 376–384.