Applied Intelligence (2023) 53:2010–2025
https://doi.org/10.1007/s10489-022-03453-z

Solving uncapacitated P-Median problem with reinforcement learning assisted by graph attention networks

Chenguang Wang[1] · **Congying Han**[1,2] · **Tiande Guo**[1,2] · **Man Ding**[1]

Accepted: 26 February 2022 / Published online: 4 May 2022

Abstract

The P-Median Problem is one of the basic cases of facility location problems and has been studied for many years. Most methods of solving it are based on classical heuristics or meta-heuristic and they don't perform well on large-scale problems according to time cost. In this paper, we propose the first reinforcement learning-based method which uses Multi-Talking-Heads Graph Attention Networks to learn representations and design a learnable attention mechanism to solve the uncapacitated P-Median Problem. We train the model using REINFORCE algorithm and show that it has good performance on uncapacitated P-Median Problem according to solution quality and time consumption. We also apply our model to the realistic dataset and empirically figure out that the difference between data distributions is one of the most important factors to influence the final performances.

Keywords P-median problem · Graph neural networks · Reinforcement learning

1 Introduction

Combinatorial Optimization (CO) is one of the most widely used subjects in real life. With the development of CO in recent 70 years, people have made breakthroughs in many related areas, such as combinatorics, operations research, and computer science, both in theory and application. In recent ten years, many traditional areas in computer science such as speech recognition, image classification, video process have achieved huge progress because of Deep Learning (DL). One reason for this is that DL can capture meaningful representations automatically which may reveal the nature of problems. Due to the great ability of DL, there is a trend on using it to solve the CO problems in recent years [1]. According to the character of CO problems, Reinforcement Learning (RL) [2] fits it well because it does not need the labels of data like in supervised learning. Otherwise, Deep Reinforcement Learning (DRL) has made great progress in many areas such as video games, robotic control which can't be solved in traditional RL. This motivates us to apply DRL to solve CO problems. Many works have been proposed to deal with classic CO problems such as Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP) and achieved acceptable results on solution quality and generalization ability.

Even though some traditional heuristic methods have achieved quite excellent performance in solution quality, most of them are limited in one or some specified problems in open datasets, generalization ability on more widely real-world problems is remaining to test. The most serious limitation of these methods is time consumption and this limit will be magnified especially when solving big batches of problems. However, once trained well, the neural networks can handle big batches of problem instances fast and get reasonable results at the same time. The results obtained by the neural networks maybe

✉ Congying Han
hancy@ucas.ac.cn

Chenguang Wang
wangchenguang19@mails.ucas.ac.cn

Tiande Guo
tdguo@ucas.ac.cn

Man Ding
dingman18@mails.ucas.ac.cn

[1] School of Mathematical Sciences, University of Chinese Academy of Sciences, No.19A Yuquan Road, Beijing, 100049, People's Republic of China

[2] Key Laboratory of Big Data Mining and Knowledge Management, Chinese Academy of Sciences, No.80 Zhonguancun East Road, Beijing, 100190, People's Republic of China

not the best, but we can take these results as good initial solutions and apply other methods to make improvements on them so that saving much time and computation cost.

In this paper, we focus on the P-Median Problem (PMP): an important special case of Facility Location Problem and has extensive application backgrounds in real life. The formal formula of PMP can be seen in Section 3.1. Same as most location problems, the complexity of solving it is NP-hard. As a consequence, many heuristic methods are proposed to solve it and most of them are based on mathematical programming or meta-heuristic.

Based on the construction scheme (introduction of schemes of solving CO problems can be seen in Section 3.2), we propose a learnable decision method and establish a model based on DRL to update the current partial solution until constructing a feasible solution. Specifically, we first handle the input PMP instances by a modified Graph Attention Networks which introduces the Multi-Talking-Heads mechanism to obtain the nodes embedding, then the attention mechanism [3] is applied to guide the sequential construction of a feasible solution. Finally, we use the REINFORCE [4], a typical RL training algorithm, to train learnable parameters in the neural networks.

To our best knowledge, we are first to solve PMP using such method and we are hoping to provide a PMP baseline based on deep learning rather than traditional heuristic methods for later researchers. We compare the performance of our model with Gurobi [5], PULP and various classical heuristic methods: Variable Neighborhood Search, Tabu Search and Genetic Algorithm in the same scale of problem instances and investigate the generalization ability on different scales. What's more, we investigate the generalization of our model on real-world dataset [6]. Experiments show that our method achieves a superior performance over classic heuristics. We also find that there are some degeneration on the performance and the main reason leading to this is the huge differences between their distributions. Overall, the contribution of this paper can be summarized as followed:

- We are first to propose a DRL-based method to solve PMP which provides a baseline for subsequent DL-based works;
- On large-scale problems, our method achieves a superior performance w.r.t. time consumption and the solution quality is still competitive when comparing other classical heuristics;
- We reveal an important insight that data distribution influences the performance of DL-based methods for CO problems and we empirically prove this through experiments.

2 Related work

Deep Reinforcement Learning (DRL) is an emerged area which incorporates Deep Learning with Reinforcement Leaning to solve problems that were previously intractable. Deep Q Networks [7] sheds the first light on DRL which scales Q-learning [8] to play video games directly from pixels. Then various classic algorithms such as REINFORCE [4], Actor-Critic [9], DPG [10] are refreshed by combining deep learning, leading to the birth of TRPO [11], DDPG [12], PPO [13], A3C [14] and many other famous DRL algorithms. These algorithms can be applied to many areas such as robotics [15], finance [16], recommendation [17] and various fields needing make decisions. Another standout work was the success of AlphaGo [18], AlphaZero [19] and Muzero [20] which are a series of works on mastering board game and video games. What's more, some theoretical results of DRL are proposed to analyse the sample efficiency [21, 22] and complexity [23]. We refer to [24–27] for a survey of DRL on various fileds.

For the development of deep learning in combinatorial optimization, Pointer Networks [28] first used attention mechanism to solve TSP supervised by example solutions which proposed the idea of solving CO problems by utilizing Recurrent Neural Networks (RNNs) to deal with variable length dictionaries. However, the demand of training labels may be unreliable especially when solving large scale problems. As a consequence, Reinforcement Learning (RL) [2] is introduced to train the deep neural networks to solve CO problems [29–32]. Lu et al. [29] proposed a framework to solve Capacitated Vehicle Routing Problem (CVRP) by using RL to select different classic improvement operators and perturbation operators. Furthermore, for graph-based CO problems, a more reasonable model, Graph Neural Networks (GNN) was used to solve CO problems [30, 33, 34]. Kool et al. [34] can be seen as a method of using GAT [35], this paper proposed a general method to solve various routing problems up to 100 cities with a single set of hyperparameters. Different from constructing feasible solution in [34], [36] proposed a reinforcement learning framework to learn the improvement heuristics for TSP and CVRP and achieved superior performance among deep learning methods both in generated data and real-world data.

In the latest research, [37, 38] focus on pre-training models and constructing heatmaps to guide the search method and [37] further generalized TSP with the scale up to 1000 cities. Closely related to our work for solving PMP, [39] proposed a method incorporate machine learning and classic solvers like SCIP [40]. Lodi et al. [39] first use machine learning method to predict the percentage of

facilities needed to change in current solutions, then add a linear constraint associated with this percentage to the Mixed Integer Programming (MIP) and run classic solvers to get resolutions. But this paper focuses on supervised learning and benefits from the powerful classic solvers. Finally, we refer to [41–43] for a comprehensive survey on existing methods and challenges on solving combinatorial optimization problems in the deep learning way.

Different from previous works which mostly focused on routing problems or solving PMP in a supervised way, we are first to design an attention-based mechanism to solve PMP and formulate PMP by designing specific state transition and reward function in the MDP framework which is significant under the paradigm of RL. Due to these improvements, we obtain a pure DRL-based model which has excellent performance on solution quality and time consumption. Especially when considering generalization ability, our method can achieve a remarkable performance w.r.t. time consumption.

3 Preliminary

3.1 P-Median problem

We here give a common description of the P-Median Problem. Given: F and C as candidate facility set and customer set, $f_j \in \mathbb{R}^+$ as fixed cost associated with constructing facility $j \in F$, $d_{i,j} \in \mathbb{R}^+$ as distance between facility $j \in F$ and customer $i \in C$, $c_{i,j} = \omega_i d_{i,j} \in \mathbb{R}^+$ as cost of shipping between candidate facility site $j \in F$ and customer location $i \in C$ and ω_i is the weight of node i as customer. The variables we need to determine are: $y_j \in \{0, 1\}$ denotes whether select (equal to 1) facility j or not (equal to 0) and $x_{i,j} \in \{0, 1\}$ determines whether there has supply received by customer $i \in C$ from facility $j \in F$. The object is to minimize the total cost of opening and operating the facilities, which can be formalized by:

$$\min \sum_{j \in F} f_j \cdot y_j + \sum_{j \in F} \sum_{i \in C} \omega_i \cdot d_{i,j} \cdot x_{i,j}.$$

Above all, we have the integer programming object as follows:

$$\begin{aligned} \min & \sum_{j \in F} f_j \cdot y_j + \sum_{j \in F} \sum_{i \in C} \omega_i \cdot d_{i,j} \cdot x_{i,j} \\ s.t. & \sum_{j \in F} x_{i,j} = 1, \forall i \in C, \\ & \sum_{j \in F} y_j = p, \\ & x_{i,j} \leq y_j, \ \forall i, j, \\ & x_{i,j}, y_j \in \{0, 1\}, \ \forall i, j. \end{aligned} \tag{1}$$

Specified in our settings, we set $f_i = 0$, $\omega_i = 1$ in unweighted case $\omega_i \in (0, 1]$ in weighted case and $F = C = \{1, 2, ..., n\}$. In the following context, we use the notation $\mathcal{I}(n, p) = (F, C, X, \omega)$ to denote a PMP instance where F is the candidate facility set, C is the customer set, $X \in \mathbb{R}^{n \times 2}$ is the two dimension euclidean coordinate matrix used for computing distance $d_{i,j}$ between two points and $\omega = (\omega_i)_i \in \mathbb{R}^n$ is the weight vector.

3.2 Scheme of solving combinatorial problems

There are two schemes of solving CO problems: the construction scheme and the improvement scheme. The first one aims to generate a feasible solution for an instance step-by-step. In each step, we update the partial solution according to some criteria and obtain a feasible solution finally. The second scheme runs above a generated feasible solution and it's promising to get a better solution than the last step by using improvement operators.

Generally speaking, the construction scheme is better than improvement scheme in terms of the Markov Decision Process (MDP). In the construction scheme, the trajectory of CO problems is finite so there are no truncation errors when using simple on-policy RL algorithms. But in the improvement scheme, we have to truncate the trajectory which will lead to truncation errors, if not, the improvement iteration can run forever.

We often choose a specific scheme based on the nature of CO problems. For some CO problems which are even hard to get a feasible solution directly, such as Capacitated Vehicle Routing Problem with Time Window (CVRP-TW), the construction scheme is a better choice. There are also CO problems that are fitted by both schemes well such as TSP.

3.3 Reinforcement learning framework

RL is a kind of methodology for the sequential decision-making problems. Specifically, by exploring the environment and taking trial-and-error searches, the agent learns to interact with the environment to maximize the long term **returns**. As is well known, RL has five key elements as described in [2]:

- $\mathcal{S}$: State space,
- $\mathcal{A}$: Action space,
- $p(s', r \mid s, a)$: State transition function,
- $r(s, a)$: Reward function,
- γ: Discount factor.

Specifically in CO problems, the state of an instance is the status of the current solution, maybe the partial solution in the construction scheme or the feasible solution in the improvement scheme. The action is the element

动作是元素集合

in the candidate set to be added to the current partial solution in construction scheme or the operator acting on the current feasible solution in the improvement scheme.

状态转移函数是已知的

The state transition function is deterministic when solving CO problems using the construction scheme. For example in TSP, when solving an instance with 10 nodes, now the partial solution contains 7 nodes and we need to select 1 node among the remaining 3 nodes, once we have chosen one specific node and added it to the partial solution, the next partial solution which contains 8 nodes is determined.

奖励是目标值的差异

Reward is usually the difference of objective function between two states to measure the improvement of taking the specific action. In the construction scheme, we can see that this kind of setting belongs to the finite-horizon task under the MDP framework, which means the interaction with the environment will terminate sooner or later. We treat the final generated feasible solution as the termination and we set discount factor $\gamma = 1$.

4 Method

图神经网络的作用是学习结点表示

In this section, we introduce Multi-Talking-Heads Graph Attention Networks, an attention-based model parameterized by Graph Neural Networks to learn node representations. Based on the learned representations, we design an attention-based facility assignment policy to fit PMP into RL framework. Then using REINFORCE [4], a classical reinforcement learning algorithm, we can train the neural networks in an end-to-end way. The paragraph arrangement in this section is as follows: we first introduce the design of facility assignment policy and the method of fitting PMP into RL in Section 4.1. And then Section 4.2 describes the main components: Encoder and Decoder in our model, which are used for learning node representations and making decisions. Finally, we show the training algorithm of solving PMP with RL in Section 4.3. The whole pipeline are demonstrated in Fig. 1.

设计策略将问题应用到RL框架

算法以端到端的方式训练

4.1 Solving PMP in the construction scheme

In this part, we formally describe the flow of solving PMP using construction scheme. Assume we want to choose p facilities among n candidate nodes, we define the partial solution in PMP formally:

Definition 1 In the uncapacitated PMP, given an instance $\mathcal{I}(n, p)$, we initialize $C_1 = \{v_1, ..., v_n\}$, $F_1 = \emptyset$ and construct a feasible solution in p steps: At step $t-1$, we have the candidate set C_{t-1} and facility set F_{t-1} and we choose $f_{t-1} := v_{t-1} \in C_{t-1}$ according to certain criteria, then update $C_t = C_{t-1} \setminus \{f_{t-1}\}$, $F_t = F_{t-1} \cup \{f_{t-1}\}$. The objective of (C_t, F_t) is given by:

$$\mathrm{OBJ}_t = \sum_{v_i \in C_t} \min_{v_j \in F_t} \{\omega_i d(v_i, v_j)\} \tag{2}$$

where ω_i is the weight of customer node v_i and $\mathrm{OBJ}_1 = 0$. We call $s_t = (C_t, F_t, \mathrm{OBJ}_t)$ the partial solution at time step t.

We aim to choose p facilities among all nodes so we have a set of partial solutions $\{s_t = (C_t, F_t, \mathrm{OBJ}_t), t = 1, ..., p\}$ through this sequential process. The final partial solution $s_p = (C_p, F_p, \mathrm{OBJ}_p)$ is the feasible solution and OBJ_p is the objective value of feasible solution.

According to Definition 1, we can fit the solving process of uncapacitated PMP into RL framework: At time step $t-1$, the state $s_{t-1} = (C_{t-1}, F_{t-1}, \mathrm{OBJ}_{t-1})$ is the partial solution of a given instance $\mathcal{I}(n, p)$, action a_{t-1} is a candidate node v_{t-1} in C_{t-1}. After taking an action, that is, updating the partial solution by adding one node, we can get a new partial solution as the next state (C_t, F_t) and the reward of time step t is $r_t = |\ \mathrm{OBJ}_{t-1} - \mathrm{OBJ}_t\ |$. A simple example for presentation can be seen in Fig. 1.

4.2 Multi-talking-heads graph attention networks

https://zhuanlan.zhihu.com/p/379722366

Attention mechanism is a famous and widely used technique in deep learning. Considering there are some technical details, we try to omit some terminologies and techniques in the implementation level and describe the high level idea in mathematics. We left the specific settings about implementation in Appendix A

Our method occupies the encoder-decoder framework. The encoder based on Graph Attention Networks (GAT) [35] is used for learning representations of node information. For learning better node representations, we introduce the Multi-Talking-Heads mechanism [44] to promote the communication between different independent Attention-Heads in Transformer module. The decoder utilizes an attention mechanism to select actions under a stochastic policy to update the current partial solution. In the following, we will specify the details of encoder and decoder respectively.

4.2.1 Encoder with multi-talking-heads mechanism

Graph Neural Networks (GNN) receive a graph $G = (V, E)$ as input, where V is the node set comprising the node information $h^{(0)} = \{h_1^{(0)}, h_2^{(0)}, ..., h_{|V|}^{(0)}\}$, $h_i^{(0)} \in \mathbb{R}^{d_0}$ and E is the edge set representing the connection relations between nodes. It outputs the final node representations $h^{(K)}$ by propagating $h^{(0)}$ after K GNN layers, each layer utilized an aggregating function Agg to get the node representations for

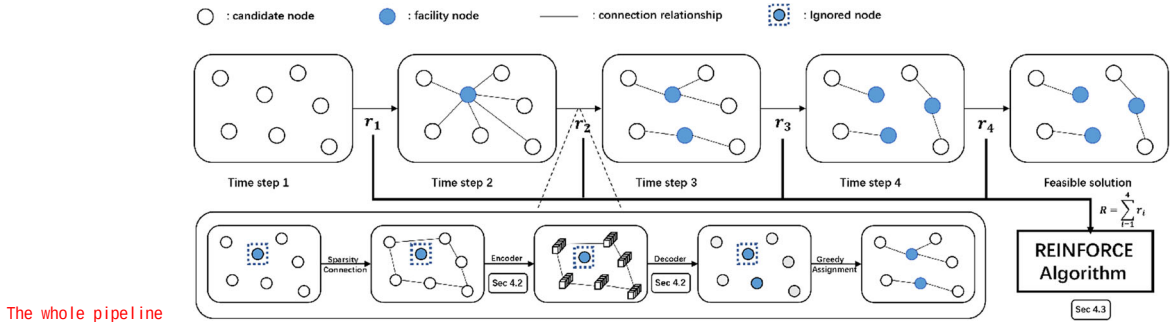


Fig. 1 For a PMP instance $\mathcal{I}(7, 3)$ which means we aim to choose 3 facilities from 7 candidate nodes. At time step 1, we have the instance to be solved as the initial state. At time step 2, we choose a facility from 7 candidate nodes according to certain criteria and connect customer nodes to their nearest facility nodes according to the cost matrix and do the same operation in time steps 2 and 3. We call solutions in time steps 1, 2, 3 and 4 the partial solution and we get the last partial solution as the feasible solution. Specifically in a transition from time step 2 to 3, we first translate the partial solution into a graph (except the node has been selected) by the sparsity connection w.r.t. the location coordinates. Then we put the graph data into GNN-based encoder to get the embedding of each node and select a facility node based on our Attention-based decoder. At last, we generate the feasible solution by greedy selection. After all, summing all rewards at each transition and updating all learnable parameters by REINFORCE algorithm

current layer:

$$h_i^{(l+1)} = \sum_{j \in N(i)} Agg(h_i^{(l)}, h_j^{(l)}) \tag{3}$$

where $i \in V$ and $N(i)$ represents the neighbors of i in graph G. Specifically in this paper, we first apply a shared linear transformation parameterized by a learnable matrix $W \in \mathbb{R}^{d_{l+1} \times d_{l+1}}$ to every node. Then self-attention [3] is performed:

$$\alpha_{i,j} = \mathbf{Att}(Wh_i^{(l)}, Wh_j^{(l)}), j \in N(i) \tag{4}$$

to obtain the importance of node j's representation to node i. Then the output of this layer is obtained by computing a linear computation of representations for each node w.r.t. this normalized attention weights after a nonlinear activation σ:

$$h_i^{(l+1)} = \sigma(\sum_{j \in N(i)} \alpha_{i,j} W h_i^{(l)}) \tag{5}$$

Now we need to specify the details about:

1. The construction of graphs in PMP. Borrow the notations in Formula (1), for a PMP instance $\mathcal{I}(n, p)$, a graph $G = (V, E)$ is constructed by: $V = F \cup C$ and connect nodes in V with graph sparsity trick to accelerate training and improve generalization [45]: for one node in V, only connecting edges with its $k\%$ nearest nodes according to the cost matrix. Then we aim to pick facility nodes $v \in F \subseteq V$ and then assign customers to facilities greedily.
2. Details about **Att** function in (4). We denote $\hat{h}_i^{(l)} = Wh_i^{(l)}$, then the classical attention mechanism is applied by mapping a query and a set of key vectors to a set of importance coefficients. The query vector is obtained by multiplying a learnable matrix $Q \in \mathbb{R}^{d_q \times d_{(l+1)}}$ left to $\hat{h}_i^{(l)}$: $q_i = Q\hat{h}_i^{(l)}$. Then set of key vectors are obtained by: $\{k_j = K\hat{h}_i^{(l)} \mid j \in N(i), K \in \mathbb{R}^{d_k \times d_{(l+1)}}\}$. After that, we can get the importance coefficients of node j to node i by:

$$A_i = \{\alpha_{i,j} \mid \alpha_{i,j} = \text{softmax}_j(< q_i, k_j >), j \in N(i)\} \tag{6}$$

Then we can take $\alpha_{i,j}$ into (5) and get the output representations.

Multi-head attention mentioned in Transformer [3] is useful to stabilize the learning process of self-attention and capture the features in different subspaces. In practice, multi-head attention is realized by executing M independent attention heads described in (4) and (5) by setting $\{W^1, W^2, ..., W^M\}$, $\{Q^1, Q^2, ..., Q^M\}$ and $\{K^1, K^2, ..., K^M\}$ and finally averaging each head's output in the following way:

$$h_i^{(l+1)} = \sigma(\frac{1}{M} \sum_{m=1}^{M} \sum_{j \in N(i)} \alpha_{i,j}^m W^m h_i^{(l)}) \tag{7}$$

During the implementation, we realize that different attention heads are independent and it's beneficial to make communications among them. As a consequence, we introduce the learnable **Multi-Talking-Heads Mechanism(MTH)** into GAT to make a comprehensive information fusion between different attention heads. Specifically, after getting each head's output representations:

$$\tilde{h}_i^{(l)} = [\sum_{j \in N(i)} \alpha_{i,j}^1 W^1 h_i^{(l)}, \sum_{j \in N(i)} \alpha_{i,j}^2 W^2 h_i^{(l)}, ..., \sum_{j \in N(i)} \alpha_{i,j}^M W^M h_i^{(l)}] \in R^{M \times d_{(l+1)}} \tag{8}$$

where $[\cdot, \cdot, ..., \cdot]$ is the horizon concatenation operator, we introduce a learnable linear transformation $T \in \mathbb{R}^{M \times M}$ to

communicate each independent attention head:

$$\bar{h}_i^{(l)} = T\tilde{h}_i^{(l)} \in \mathbb{R}^{M \times d_{(l+1)}} \tag{9}$$

and then get the output:

$$h_i^{(l+1)} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M} \bar{h}_{i,m}^{(l)}\right) \tag{10}$$

where $\bar{h}_{i,m}^{(l)} = \sum_{j \in N(i)} \alpha_{i,j}^m W^m h_i^{(l)}$.

Generally speaking, we first construct a graph based on given PMP instances and get the initial representations $\{h_i^{(0)} \in \mathbb{R}^{d_0} \mid i = 1, 2, ..., \mid V \mid\}$ by embedding each original node information $\{x_i \in \mathbb{R}^{d_x} \mid i = 1, 2, ..., \mid V \mid\}$ through a transformation f_θ which is parameterized by the neural networks:

$$h_i^{(0)} = f_\theta(x_i) \in \mathbb{R}^{d_0}, i = 1, 2, ..., \mid V \mid .$$

After that, we get node representations $\{h_i^{(K)} \in \mathbb{R}^{d_K} \mid i = 1, 2, ..., \mid V \mid\}$ by implementing **Multi-Talking-Heads Mechanism(MTH)** on these initial node representations for K times and get node representations $\{h_i^{(K)} \in \mathbb{R}^{d_K} \mid i = 1, 2, ..., \mid V \mid\}$. The overview of these operations can be seen in Fig. 2.

4.2.2 Decoder with attention mechanism

Following the construction scheme of solving CO problems, we design a learnable attention-based decoding mechanism by choosing the facility and then assign the customers to facilities greedily to construct the feasible solution.

Decoding is executed sequentially, and at time step $t = 1, 2, ..., \mid F \mid$, the decoder outputs the facility f_t based on node representations obtained from encoder and the state of current partial solution. During the decoding process of time step t, we have the partial solution $s_t = (C_t, F_t, OBJ_t)$ defined in 1 and want to choose one node $f_t \in C_t$ which will be added to F_t. The idea of our learnable decoding strategy is: Setting a context vector c which contains the information of partial solution s_t and computing a stochastic strategy to provide a guideline for choosing candidate facility by assigning probabilities to each node in C_t and giving high probability on the nodes that have low compatibility, which means nodes with low compatibility are more intended to be removed from C_t. The illustration of decoding steps is shown in Fig. 3.

Context vector comprises the information of current partial solution s_t and it can guide us to choose one facility from the candidate set at each decoding step. Specifically, the context vector defines as follows:

$$\mathbf{h}_c^{(K)} = \frac{1}{\mid V \mid -t+1}\sum_{i \in C_t} h_i^{(K)} \tag{11}$$

which means the average of node representations except for those of facilities have been chosen. We then compute a new context vector $\mathbf{h}_c^{(K+1)}$ using MTH described before. After that, we design a decoding strategy based attention mechanism to select facility. Similarly we compute a single query $\mathbf{q_c}$ from the context vector $\mathbf{h}_c^{(K+1)}$ and keys from the node representations $h_i^{(K)}$:

$$\mathbf{q_c} = W^Q \mathbf{h}_c^{(K+1)} \quad \mathbf{k}_i = W^K h_i^{(K)}. \tag{12}$$

We compute the compatibility between the query and all candidate nodes which can be chosen at time t:

$$u_{(c)j} = \begin{cases} \frac{\mathbf{q_c}^T \mathbf{k}_j}{\sqrt{d_k}} & \text{if node } j \text{ is in } C_{t-1} \\ -\infty & \text{otherwise.} \end{cases} \tag{13}$$

Here $d_k = \frac{d_K}{M}$ is the query/key dimensionality and M is the number of attention heads in **Multi-Talking-Heads Mechanism(MTH)**.

Computation of stochastic strategy At time step t, we need to choose a facility from the current candidate set obeying a

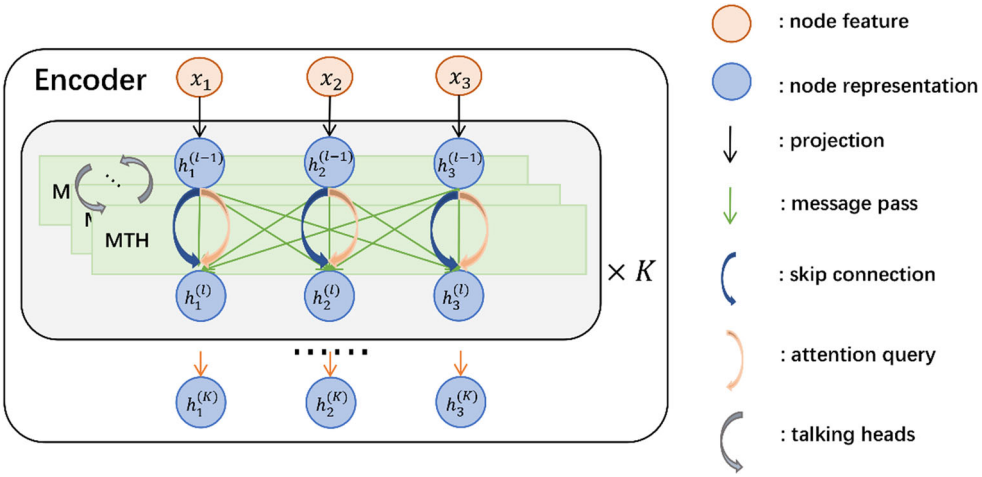


Fig. 2 Transformer-based encoder. We first embed node information and process the initial embeddings by K Transformer modules according to a given adjacency matrix, each consists of multi-head attention(MHA), talking-heads operator, and feed-forward (FF) sublayer. Finally, we get node embeddings

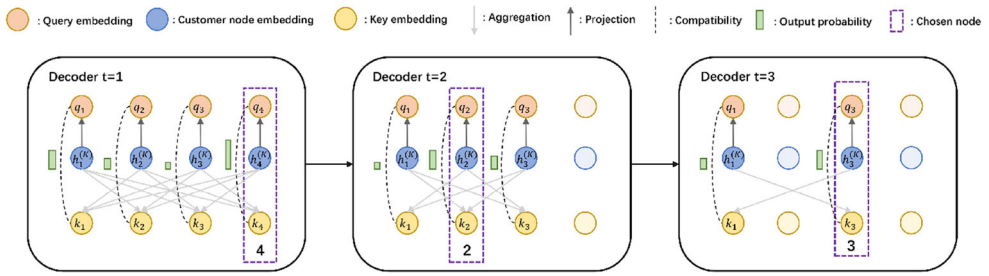


Fig. 3 Attention-based decoder. Given the node representations obtained from encoder, we aim to choose facilities from these nodes according to the proposed attention mechanism. At each time step t, our decoder selects the least relevant out of all nodes as the facility node. The example shows how the facility nodes (4, 2, 3) are chosen

stochastic strategy $p_\theta(f_t \mid s_{t-1})$. We add a final layer with a single attention head (M=1 so $d_k = d_K$) and only compute the compatibilities $u_{(c)j}$ using (13). Similarly in Attention Model [34], we clip the results with $[-C, C]$ (C=10) using tanh function:

$$u_{(c)i} = \begin{cases} C \cdot \tanh\left(-\frac{\mathbf{q}_{(c)}^T \mathbf{k}_i}{\sqrt{d_k}}\right) & \text{if node } v_i \text{ is in } C_{t-1} \\ -\infty & \text{otherwise.} \end{cases} \tag{14}$$

Finally we compute the probability using softmax function:

$$p_t^i = p_\theta(v_i \mid s_{t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}}. \tag{15}$$

and get the stochastic strategy $\boldsymbol{p}_t = (p_t^1, p_t^2, ..., p_t^{|C_{t-1}|})$

4.3 Algorithm

As described in Definition 1 and Section 4.2.2, given an initialization s_0 with $C_1 = \{v_1, ..., v_n\}$, $F_1 = \emptyset$ and $\text{OBJ}_1 = 0$, we can obtain the probability distribution $\mathbf{P}(\boldsymbol{a} \mid s_1)$ over C_1 and pick action a_1 corresponding to a node $f_1 = v_i \in C_1$ with probability p_1. Then updating partial solution with $C_2 = C_1 \setminus \{f_1\}$, $F_2 = F_1 \cup \{f_1\}$ and the state transfers to $s_2 = (C_2, F_2, \text{OBJ}_2)$. Similarly, we do this until generating a feasible solution (C_p, F_p, OBJ_p). We parameterize the obtained policy by the neural networks and denote it as $\boldsymbol{\pi}_\theta(a \mid s), \forall s$. For an episode of length p, the policy $\boldsymbol{\pi}_\theta(a \mid s)$ induce the trajectory distribution:

$$p_\theta(\boldsymbol{\tau}) = \mathbb{E}_{s_1 \sim p(s_1)} \prod_{t=1}^{p} \boldsymbol{\pi}_\theta(a_t \mid s_t)$$

because the state transition is deterministic in CO problems, where $\boldsymbol{\pi}_\theta(a_t \mid s_t) = \boldsymbol{p}_t$ as defined in (15). We use REINFORCE [4] with baseline b which is independent of states and actions to maximize the cumulative rewards by policy gradient theorem:

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[(L(\boldsymbol{\tau}) - b) \nabla_\theta \log(\prod_{t=1}^{p} \pi_\theta(a_t \mid s_t)) \right]. \tag{16}$$

where $L(\boldsymbol{\tau})$ is the cumulative rewards of trajectory τ:

$$L(\tau) = \sum_{t=1}^{p} r_t = \sum_{t=1}^{p} \mid \text{OBJ}_t - \text{OBJ}_{t+1} \mid . \tag{17}$$

The whole pipeline of solving PMP is described in Algorithm 1.

Algorithm 1 Solving PMP with REINFORCE

Input: PMP instances training set $\mathcal{D} = \{\mathcal{I}(n, p) = (F, C, X, \omega)\}$

Initialization: Neural networks parameters $\theta = (\theta_E, \theta_D)$ for Encoder and Decoder denoted as $f_\theta = (f_{\theta_E}, f_{\theta_D})$.

for many epochs **do**
  for batch in $\mathcal{D}$ **do**
    Set $F_1 = \emptyset$, $C_1 = C$, $OBJ_1 = 0$
    if $\mathcal{I}(n, p)$ is unweighted **then**
      Get node representation $h^{(K)} = f_{\theta_E}(X)$
    else
      Get node representation $h^{(K)} = f_{\theta_E}(X, w)$
    end if
    for $t = 1, 2, ..., p$ **do**
      Computing context vector $\mathbf{h}_c^{(K+1)}$ based on $\mathbf{h}_c^{(K)}$ in (11)
      Obtaining stochastic policy $\boldsymbol{p}_t$ by (14) and (15) with f_{θ_D}
      Sampling $f_t := v_i$ with p_t^i from C_t w.r.t. distribution $\boldsymbol{p}_t$
      Updating partial solution: $C_{t+1} = C_t \setminus \{f_t\}$, $F_{t+1} = F_t \cup \{f_t\}$
      and OBJ_{t+1} by (2), computing reward $r_t = \mid \text{OBJ}_{t+1} - \text{OBJ}_t \mid$
    end for
    Computing gradient in (2) by collected $\{p_t^i\}$ and $\{r_t\}$
  end for
  Update parameters θ in neural networks
end for

5 Experiments

In this section, we report our experimental results on uncapacitated PMP. We first train our model on the scale of $(n, p) = (20, 4), (50, 8), (100, 15)$ and show the performance on the same scale during training. What's more, we investigate our model's generalization ability on the real-world dataset. Considering that there are no pure DL based methods as far as we know, we compare the performance with various classical heuristics and results reveal that our method achieves competitive performance both on generated data and real-world data.

5.1 Settings

5.1.1 Data normalization

From (1), we denote $d_{i,j} = || e_i - e_j ||_2$ where $e_i = (x_1^i, x_2^i)$ represents a two dimension coordinate for node i, $x_1^{min} = \min_i\{x_1^i\}$, $x_2^{max} = \max_i\{x_2^i\}$, $\omega_{max} = \max_i\{\omega_i\}$ and $||\cdot||_2$ is the l_2 norm. To keep the data in the same scale, we normalize the input data to [0, 1] by:

$$\tilde{e}_i = \left(\frac{x_1^i - x_1^{min}}{x_1^{max} - x_1^{min}}, \frac{x_2^i - x_2^{min}}{x_2^{max} - x_2^{min}}\right), \quad \tilde{\omega}_i = \frac{\omega_i}{\omega_{max}}.$$

In this normalizing way, we have our objective function:

$$\begin{aligned} &\text{argmin}_{x_{i,j}} \sum_{j\in F}\sum_{i\in C} \omega_i \cdot d_{i,j} \cdot x_{i,j} \\ &= \text{argmin}_{x_{i,j}} \sum_{j\in F}\sum_{i\in C} \frac{\omega_i}{\omega_{max}} \cdot \frac{||\,(x_1^i - x_1^j, x_2^i - x_2^j)\,||_2}{d_Z} \cdot x_{i,j} \\ &= \text{argmin}_{x_{i,j}} \sum_{j\in F}\sum_{i\in C} \tilde{\omega}_i \cdot ||\,\tilde{e}_i - \tilde{e}_j\,||_2 \cdot x_{i,j} \end{aligned} \tag{18}$$

where $d_Z = ||\,(x_1^{max} - x_1^{min}, x_2^{max} - x_2^{min})\,||_2$. The first equation holds because ω_{max} and d_Z are constants and division by a constant factor has no impact on the 'argmin' function when the contraints remain same. The second equation holds because

$$\begin{aligned} &\frac{||\,(x_1^i - x_1^j, x_2^i - x_2^j)\,||_2}{d_Z} \\ =&\frac{||\,((x_1^i - x_1^{min}) - (x_1^j - x_1^{min}), (x_2^i - x_2^{min}) - (x_2^j - x_2^{min}))\,||_2}{d_Z} \\ =&||\,\tilde{e}_i - \tilde{e}_j\,||_2\,. \end{aligned} \tag{19}$$

Finally, we complete the normalization ensuring the solution $\{x_{i,j}, i \in C, j \in F\}$ invariant at the same time.

5.1.2 Data generation

During training period, we generate data by sampling two-dimension coordinates (x_1, x_2) from $[0, 1] \times [0, 1]$ and weights ω from [0, 1] uniformly on the fly at each epoch. During testing period, we first show the performance on the generated data sampled from uniform distribution. We also investigate the performance on real-world data from [6].

[6] provides a publicly available locational dataset which covers entire area of Slovakia and consists of the graph of the road network and almost 700,000 connected demand points. Thanks to this work, we can get various modes of real-world data by sampling points from the dataset which provides latitude, longitude, and weights of locations. As a consequence, we translate these geometry information into two-dimension coordinates using equirectangular projection:

$$\begin{aligned} x &= R\,(\lambda - \lambda_0)\cos\varphi_1, \\ y &= R\,(\varphi - \varphi_0)\,, \end{aligned} \tag{20}$$

where

- λ is the longitude of the location;
- φ is the latitude of the location;
- λ_0 is the central parallel of the map, here we take it as the center longitude of all data;
- φ_0 is the central parallel of the map, here we take it as the center latitude of all data;
- φ_1 are the standard parallels (north and south of the equator) where the scale of the projection is true, here we take $\varphi_1 = \varphi_0$;
- R is the radius of the globe, here we take $R = 6357$;
- x is the horizontal coordinate of the projected location on the map;
- y is the vertical coordinate of the projected location on the map.

After implementing the projection, we normalize these data to [0, 1] by the method described in (18).

5.1.3 Hyperparameters

For unweighted PMP, node information only contains two-dimension coordinates (x_1, x_2), and for weighted PMP, each node contains three-dimension information: (x_1, x_2, w) where w is the weight coefficient as a customer. We train 500 epochs and process 500 batches of 256 instances generated on the fly at each epoch.

For the configurations about encoder and decoder, number of attention heads and Transformer modules are $M = 4, K = 5$ respectively, graph sparsity coefficient $k = 75$ for $(20, 4), (50, 8)$ and $k = 20$ for $(100, 15)$. We use a decay learning rate $\eta = 10^{-4}$ with a decay factor of 0.98. As for the rollout baseline, we use an exponential baseline during the first epoch which is a practical method to stabilize the training. All experiments during the training period run in TESLA-V100 16G. Specific details are listed in Table 1 below.

Table 1 Model Configuration

Module	DESCRIPTION
Parameters Initialization	Uniform$(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$
Hidden Layers Dimension	dim=128 with ReLU activation
Num. of Attention Heads	$M = 4$
Num. of Transformer Modules	$K = 5$
Graph Sparsity	$k = 75$ for (20, 4), (50, 8), $k = 20$ for (100, 15)
Optimizer	Adam [46] with lr=1e-4, lr_decay=0.98

5.1.4 Decoding strategy and baselines

After finishing the training of our model, we obtain a stochastic policy due to using REINFORCE algorithm [4], so we sample 1280 solutions for an instance and report the best. More sampling improves solution quality at increased computation. We report the possible 'best solution' obtained by Gurobi [1] as the ground-truth. Performance of each method is evaluated by the optimality gap (%) which is calculated by:

$$\text{gap} = \left(\frac{\text{object value obtained by testing method}}{\text{object value obtained by Gurobi}} - 1\right) \times 100\%$$

and its standard deviation which denotes 'Std' in the Table.

5.1.5 Running time

time consumption is also an important factor during comparing different methods. However, time consumption is hard to compare because the difference of implementation (Python or C++) and hardware (GPU and CPU). In this paper, we take a practical view and report the average time it takes on solving an instance from a test set of 10000 instances, either on a single GPU (RTX 3090) or on a CPU (Intel(R) Core(TM) i9-10900KF CPU).

5.2 Results on random generated data

We report the optimal results by Gurobi, as well as by PULP, a free open source software written in Python used to describe optimisation problems as mathematical models. Given that there are no pure DL based methods, we compare our model with some classical heuristics: Variable Neighborhood Search (VNS) and Tabu Search (TS), Genetic Algorithm (GA).

We first compare the performance on problem scale: (20, 4), (50, 8), (100, 15) in Table 2. Our model is trained on random generated data and we compare these methods in the same distribution. We can see that our method achieves the best performance among all heuristics by a big margin when the problem scale remains same during training and testing. Additionally, our methods achieves the minimum standard deviation w.r.t. the optimality gap from Table 2, which means our method is the most stable one among all heuristic methods. The performance of our model on various problem scales are shown in Fig. 4a[1]. The generalization results are stable when the problem scale under 500 nodes but the performance begins to degenerate gradually on large scale datasets $(n, p) = (500, 15), (1000, 15)$. This is reasonable for a DL based method but the solution quality is still competitive.

As for the time consumption shown in Fig. 4b, when applying to large scale problem $(n, p) = (500, 15), (1000, 15)$, our model runs faster than other methods by a big margin. In particular, when solving an instance with problem scale $(n, p) = (1000, 15)$, Gurobi, GA, TS and VNS take 252.9s, 105.52s, 279.96s, 41.34s respectively while our method only takes 3.44s in average. And in relatively small scales, our method also achieves the competitive performance comparing with other methods. Associated with the results in Fig. 4a, even though the our model's optimality gap on $(n, p) = (500, 15), (1000, 15)$ are 6.32% and 11.43% which are bigger than TS's to a small extent, our model's charming performance on time consumption can offset this small flaw with no doubt.

5.3 Results on real-world data

In this part, we use the model trained on randomly generated data to investigate performance and generalization ability on real-world data. Similarly, we first solve these transformed real-world data by Gurobi and compare them with results obtained by our model.

We first compare the performance of our model with classical heuristics in Table 3. In our implementation, when the problem scale during test period remains same as that during training, our model is relatively stable even though there is a huge gap between training and testing distribution and achieves the best results among all heuristic methods. Similarly in the case of real-world data, our methods is also the most stable one among all heuristic methods. However, according to Fig. 5, the performance is not as competitive as in generated data when employing our model on large scale instances generated from real-world dataset. But given the excellent performance on time consumption,

[1] The performance of VNS is too bad to demonstrate

Table 2 Our model vs baselines on generated data. The gap % is w.r.t. the best value across all methods. Std means the standard deviation(%) of the gap. Results in boldface type mean the best

		$(n, p) = (20, 4)$				$(n, p) = (50, 8)$				$(n, p) = (100, 15)$			
	Method	Obj.	Gap	Std	Time	Obj.	Gap	Std	Time	Obj.	Gap	Std	Time
Unweighted	Gurobi	2.97	0.00%	-	0.01s	5.32	0.00%	-	0.15s	7.68	0.00%	-	0.65s
	PULP	2.97	0.00%	-	0.03s	5.32	0.00%	-	0.33s	7.68	0.00%	-	1.33s
	VNS	4.10	38.89%	23.55%	0.34s	7.65	43.67%	18.45%	1.26s	11.21	46.00%	14.37%	3.92s
	TS	3.03	2.07%	1.39%	0.01s	5.49	3.21%	2.28%	0.05s	7.96	3.67%	1.83%	0.33s
	GA	2.99	0.67%	0.52%	0.11s	5.43	2.01%	1.89%	0.52s	8.10	5.46%	2.37%	1.49s
	Ours	**2.97**	**0.08%**	**0.29%**	0.01s	**5.33**	**0.28%**	**0.45%**	0.12s	**7.77**	**1.21%**	**0.80%**	0.22s
Weighted	Gurobi	1.34	0.00%	-	0.03s	2.44	0.00%	-	0.14s	3.54	0.00%	-	0.66s
	PULP	1.34	0.01%	-	0.04s	2.44	0.00%	-	0.24s	3.54	0.01%	-	1.36s
	VNS	2.00	49.25%	22.97%	0.41s	3.68	50.81%	19.87%	1.23s	5.42	53.10%	16.65%	4.25s
	TS	1.36	1.95%	1.13%	0.01s	2.51	3.04%	2.54%	0.19s	3.67	3.61%	2.04%	1.12s
	GA	1.35	1.21%	0.78%	0.19s	2.48	1.64%	1.15%	0.72s	3.74	5.65%	2.56%	2.08s
	Ours	**1.34**	**0.09%**	**0.28%**	0.06s	**2.47**	**1.23%**	**1.18%**	0.13s	**3.64**	**2.72%**	**1.44%**	0.20s

the degeneration on solution quality is still within the arrange of tolerance. We further investigate the potential factors of this degeneration in Section 6.2.

6 Discussion

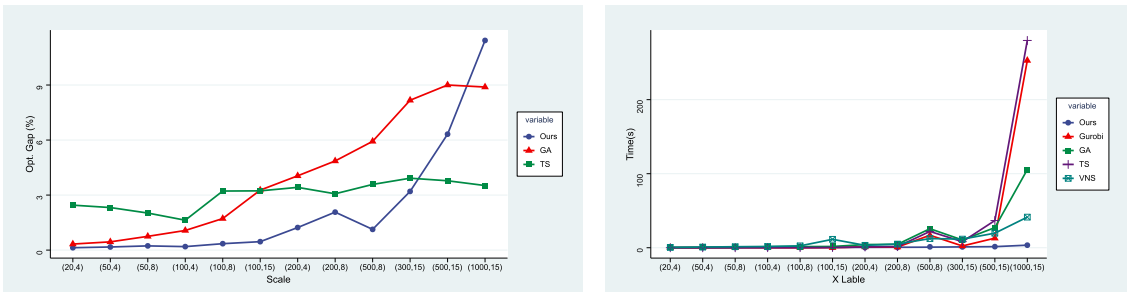
In this section, we mainly discuss the generalization ability of our model by testing in different instance scales with respect to the number of nodes n and facilities p. We further inspect the potential factor of influencing the generalization ability and figure out the difference of data distribution between training and testing plays an important role on the final results.

6.1 Generalization with respect to *n* and *p*

About the presentation of Fig. 6, we investigate the generalization ability of model trained on $(n, p) = (20, 4)$, $(50, 8)$ and $(100, 15)$ on two aspects:

1. test on (n_i, p_i), where $n_i \neq n$, $p_i = p$ in Fig. 6a, b, c;
2. test on (n_i, p_i), where $n_i = n$, $p_i \neq p$ Fig. 6d e, f.

Generally speaking, our model achieves the best performance when the scale during training is same as that during testing. For the generalization on n, as shows in Fig. 6a, b, c, the performance degenerates to some extent but is still within the range of tolerance. However, we can see that the generalization ability of our model is



(a) Comparisons of optimal gap(%) on various problem scales

(b) Comparisons of time consumption on various problem scales for solving an instance in average

Fig. 4 Comparisons on optimal gap and time cost on various problem scales on generated data. In Fig. 4a, we can see our method achieves the best performance according to the solution quality and competitive results on large scales, e.g. $(n, p) = (500, 15)$, $(1000, 15)$. Considering the time cost in Fig. 4b, our method achieves the dominate performance on large scale problems. Specifically on problem scale $(n, p) = (1000, 15)$, Gurobi, GA, TS and VNS take 252.9s, 105.52s, 279.96s, 41.34s respectively while our method only takes 3.44s

Table 3 Our model vs baselines on real data. The gap % is w.r.t. the best value across all methods. Std means the standard deviation(%) of the gap. Results in boldface type mean the best

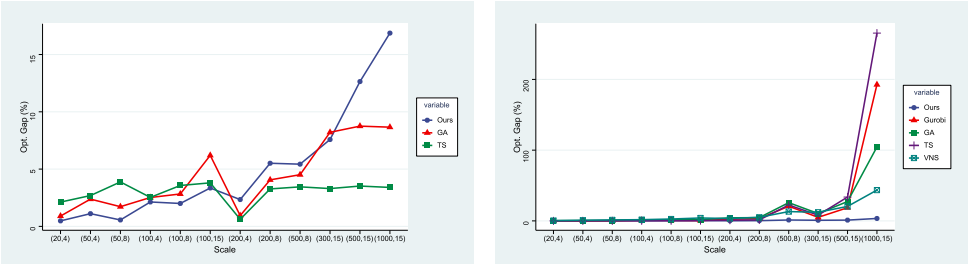
	$(n, p) = (20, 4)$				$(n, p) = (50, 8)$				$(n, p) = (100, 15)$			
Method	Obj.	Gap	Std	Time	Obj.	Gap	Std	Time	Obj.	Gap	Std	Time
Gurobi	2.21	0.00%	-	0.03s	3.92	0.00%	-	0.20s	5.60	0.00%	-	0.31s
PULP	2.21	0.06%	-	0.04s	3.92	0.05%	-	0.27s	5.60	0.00%	-	1.32s
VNS	3.07	38.91%	20.88%	0.44s	5.72	45.69%	21.02%	1.32s	8.19	46.25%	18.37%	4.01s
TS	2.25	1.58%	1.38%	0.01s	4.05	3.29%	2.32%	0.05s	5.80	3.51%	1.80%	0.32s
GA	2.24	1.34%	0.86%	0.12s	4.02	2.55%	1.53%	0.52s	5.97	6.61%	3.18%	1.61s
Ours	**2.22**	**0.42%**	**1.56%**	0.06s	**3.94**	**0.51%**	**0.53%**	0.10s	**5.70**	**1.78%**	**1.62%**	0.18s

more sensitive to the number of facility p than the node number n. We speculate that this is associated with our decoder strategy: for fixed (n, p), our model is trained to decode p steps to try to get better solutions, if we test it on different scales of instances w.r.t. p, it violates the potentiality of our model to solve them in p steps. Relatively speaking, generalization ability associated with n is determined by the effects of representation learned by the encoder which shows good performance on many tasks.

6.2 Influence of data distribution

According to the experimental results, we find the difference between training and testing distribution is one of the most key factors that affect the final performance and some data can be simulated by randomly generating such as coordinates, but some data can't such as weights of customers in PMP. Maybe that's why many deep learning models work well in TSP because it only needs to generate two-dimension coordinates in TSP tasks. When we investigate the generalization of the deep learning model, except considering the ability of scalability on instance size, we should pay attention to the distribution of real-world data as well.

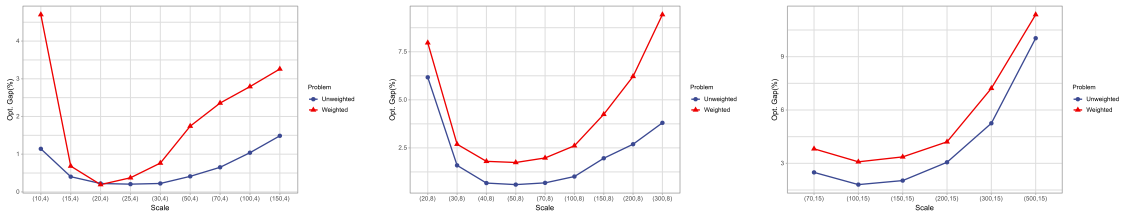
Just as Fig. 7a shows, the cumulative distribution functions (CDF) of weights between two distributions are quite different, which indicates the specific weights distribution of customer nodes in real-world data is intrinsically different from those of randomly generated. Moreover, using t-SNE [47], we visualize the high dimensional representations learned by the encoder in Fig. 7 and we can see that in Fig. 7b, the representations learned from uniform data can not cover the real data even though



(a) Comparisons of optimality gap(%) on various problem scales

(b) Comparisons of time consumption on various problem scales for solving an instance in average

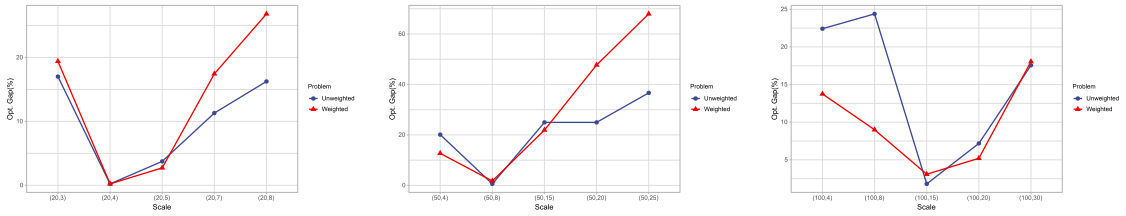
Fig. 5 Comparisons on optimality gap and time cost on various problem scales on real-world data. In Fig. 5a, we can see our method achieves the best performance according to the solution quality on relatively small problem scale and competitive results within the arrange of tolerance. Considering the time cost in Fig. 5b, our method achieves the dominate performance on large scale problems. Specifically on problem scale $(n, p) = (1000, 15)$, Gurobi, GA, TS and VNS take 192.61s, 104.49s, 265.47s,43.54s respectively while our method only takes 3.45s



(a) Generalization results of model trained on $(n, p) = (20, 4)$ w.r.t. n

(b) Generalization results of model trained on $(n, p) = (50, 8)$ w.r.t. n

(c) Generalization results of model trained on $(n, p) = (100, 15)$ w.r.t. n



(d) Generalization results of model trained on $(n, p) = (20, 4)$ w.r.t. p

(e) Generalization results of model trained on $(n, p) = (50, 8)$ w.r.t. p

(f) Generalization results of model trained on $(n, p) = (100, 15)$ w.r.t. p

Fig. 6 Generalization results w.r.t. n and p on generated data. Fig. 6a, 6b, 6c show the generalization performance w.r.t. n and Fig. 6d, 6e, 6f show that w.r.t. p

the amount of uniform data is one hundred times more than real data. However, in Fig. 7c and 7d, the distribution of the same kind of data is compact and clustered.

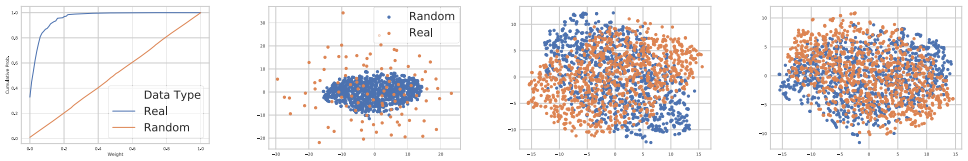
6.3 Stability of our model

We will show the stability of our model in two aspects:

1. Stability during the training period: we train our model under 5 randomly generated seeds on generated data and test it with respect to the optimal gap at the end of each epoch;
2. Stability during the testing period: We randomly generate 5 seeds and 5 datasets and test our model which is trained on different seeds according to the optimal gap on different datasets.

We do stability tests both on the unweighted and weighted case and results of stability during the training period are shown in Fig. 8. We can see that the optimal gap on the testing set which is generated on each epoch continues to improve steadily over time and different random seeds have small impacts on it .

Results of stability during the testing period are shown in Fig. 9. We test our model on 5 randomly generated datasets and each dataset contains 1280 instances. Each color curve represents the performance of a random seed on 5 datasets and the shaded part of each color curve represents the



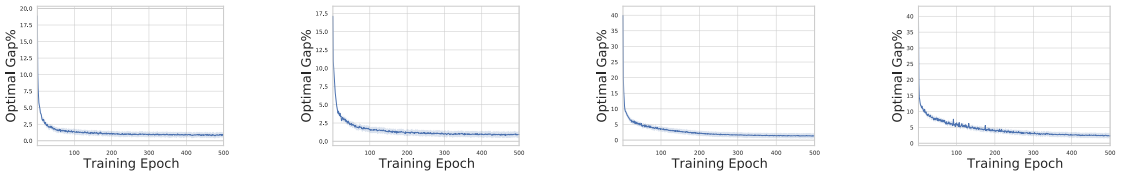
(a) CDF of normalized data in Partizanske dataset (blue) and uniform distribution data (orange)

(b) representation visualization of random data (blue) and real data (orange)

(c) representation visualization of two sets of real data

(d) embedding visualization of two sets of random data

Fig. 7 (a): After normalizing real-world data to $[0, 1]$, the blue line indicates the cumulative distribution function (CDF) of weights data in the Partizanske dataset and the orange one indicates the CDF of uniform distribution which is used to train our model; (b): orange: 2000 real data, blue: 2000×100 random data; (c): two sets of 20000 real data; (d): two sets of 20000 real data



(a) Training Fig of optimality gap at each epoch in the unweighted case for $(n, p) = (20, 4)$

(b) Training Fig of optimality gap at each epoch in the weighted case for $(n, p) = (20, 4)$

(c) Training Fig of optimality gap at each epoch in the unweighted case for $(n, p) = (50, 8)$

(d) Training Fig of optimal gap at each epoch in the weighted case for $(n, p) = (50, 8)$

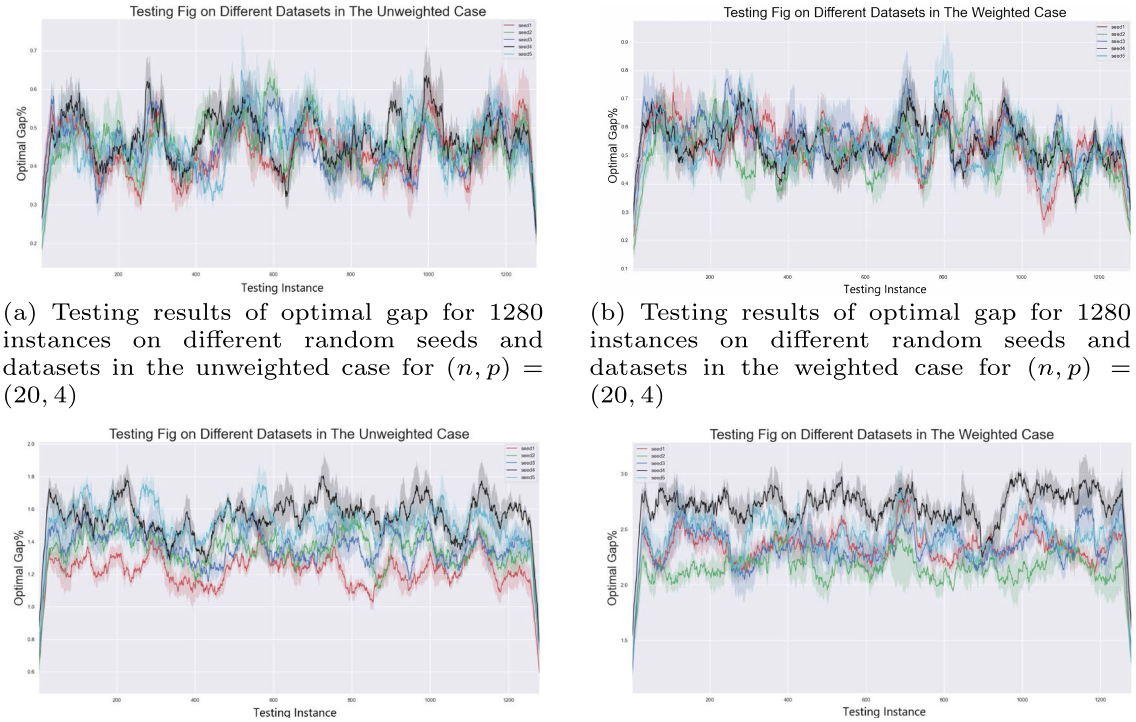
Fig. 8 Training stability according to the optimal gap% tested at each epoch

variance of each random seed on the 5 datasets. We can see that performance of our model is robust w.r.t. different random seeds and the datasets during the testing period.

7 Conclusion and future work

In this paper, we propose a method that have good performance and generalization ability on uncapacitated PMP. Different from the improvement scheme, MDP of CO problems solved with construction scheme is often seen as a finite horizon task in RL, so our model can solve the uncapacitated PMP in fixed steps. Otherwise, we provide intriguing insights on the generalization ability of our model: the number of facility p and distribution of training dataset are main reasons to affect generalization results.

One significant research as an extending of ours is how to design an adaptive decoder strategy which has good generalization on different facility number p. Otherwise, a closely related problem is capacitated PMP which is harder than uncapacitated. Same as described in Section 6.2, we can't simulate facility capacity by generating randomly like coordinates, one critical reason is that it will lead to



(a) Testing results of optimal gap for 1280 instances on different random seeds and datasets in the unweighted case for $(n, p) = (20, 4)$

(b) Testing results of optimal gap for 1280 instances on different random seeds and datasets in the weighted case for $(n, p) = (20, 4)$

(c) Testing results of optimal gap for 1280 instances on different random seeds and datasets in the unweighted case for $(n, p) = (50, 8)$

(d) Testing results of optimal gap on different random seeds and datasets in the weighted case for $(n, p) = (50, 8)$

Fig. 9 Testing stability according to the optimal gap% on different test sets contains 1280 randomly generated instances

no feasible solutions. Even though we generate solvable instances, the distribution of generated capacity may not suit real-world data. So how to generate data that has specific practical significance is the key to solve the capacitated PMP by deep learning.

Appendix A: Implementation Details of Transformer Module

Attention mechanism is embedded in the Transformer [3]. In this section we present some technical details about the implementation. For a node within a graph, the attention means the weights of messages should receive from its neighbor nodes. This node's weight of messages depends on the compatibility between its query and the key of its neighbors. Formally, we denote the node representation as $h_i \in \mathbb{R}^{d_h}$ and matrices $W^K, W^Q \in \mathbb{R}^{d_k \times d_h}$, $W^V \in \mathbb{R}^{d_v \times d_h}$ and compute the key $k_i \in \mathbb{R}^{d_k}$, value $v_i \in \mathbb{R}^{d_v}$, query $q_i \in \mathbb{R}^{d_k}$ for each node in the following way:

$$q_i = W^Q h_i,\ k_i = W^K h_i,\ v_i = W^V h_i, \tag{21}$$

From these queries and keys, we can obtain the compatibility of node i's query and node j's key: $u_{ij} \in \mathbb{R}$ as the scaled dot-product [3]:

$$u_{ij} = \begin{cases} \frac{q_i{}^T k_j}{\sqrt{d_k}} & \text{if node } i \text{ is adjacent to } j \\ -\infty & \text{otherwise.} \end{cases} \tag{22}$$

We then compute the attention weights $a_{ij} \in [0, 1]$ using softmax function:

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{ij'}}} \tag{23}$$

At last, we complete the information aggregation to get a new node representation by:

$$h'_i = \sum_j a_{ij} v_j \tag{24}$$

Furthermore, as noted in [3], it's beneficial to introduce **Multi-head Attention** which has multiple attention heads which can help to capture different kinds of information from different neighbors. In particular, we parameterize a set of matrices $\{W_m^K, W_m^Q \in \mathbb{R}^{d_k \times d_h}, W_m^V \in \mathbb{R}^{d_v \times d_h}, m = 1, 2, ...M\}$ and obtain M node representations for node i by above attention mechanism $\{h'_{i,m}, m = 1, 2, ...M\}$. The final multi-head attention value for node i is obtained by:

$$MHA_i(h_1, ..., h_n) = \sum_{m=1}^{M} W_m^O h'_{i,m} \tag{25}$$

where $\{W_m^O \in \mathbb{R}^{d_h \times d_v}, m = 1, 2, ...M\}$

At the implementation level, we need some extra neural networks modules to obtain these representations: Feed-forward sublayer and batch normalization [48].

The feed-forward sublayer computes each node's representations using a linear projection followed a ReLU activation:

$$FF(h_i) = W^{ff,1} \cdot ReLU(W^{ff,0} h_i + b^{ff,0}) + b^{ff,1} \tag{26}$$

The batch normalization aims to reduce internal covariate shift during training neural networks and the formula with a batch of inputs $\{x^{(k),k=1,2,\cdots}\}$ is as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$

The overall pipeline of this process is shown in Fig. 10

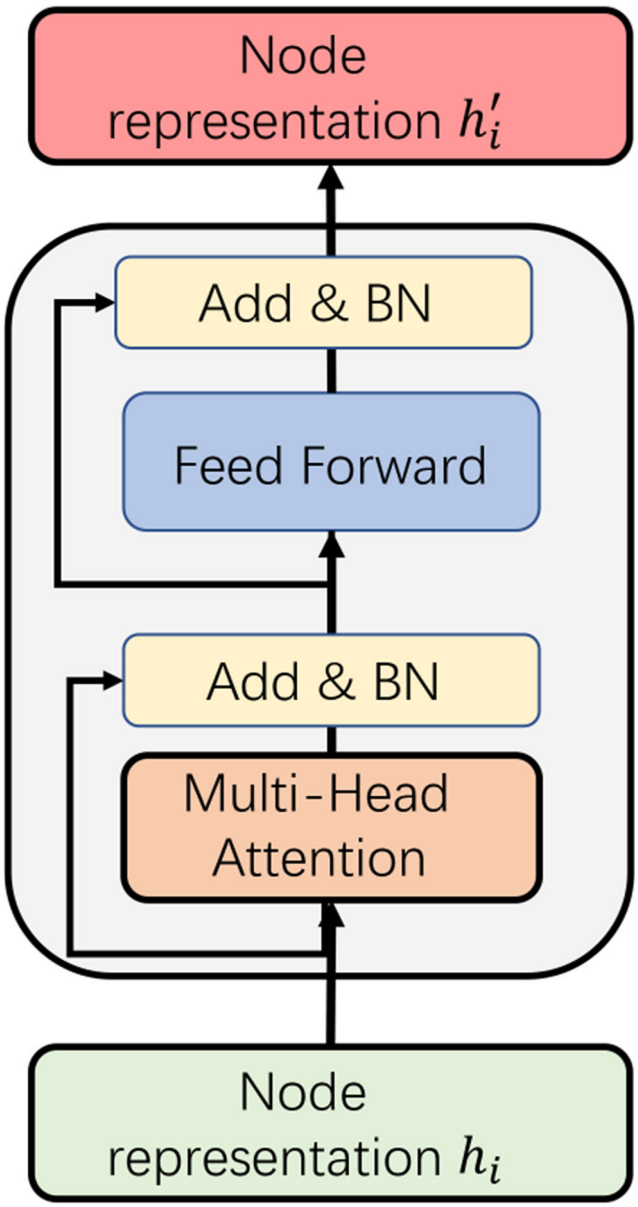


Fig. 10 The pipeline of Transformer module

Acknowledgements This paper is supported by National key research and development program of China (2021YFA1000403) and supported by the National Natural Science Foundation of China (Nos. 11991022, U19B2040) and the Fundamental Research Funds for the Central Universities.

References

1. Guo T, Han C, Tang S, Ding M (2019) Solving combinatorial problems with machine learning methods. In: Nonlinear Combinatorial Optimization. Springer, pp 207–229
2. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press
3. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, pp 5998–6008
4. Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8(3-4):229–256
5. Gurobi Optimization LLC (2021) Gurobi Optimizer Reference Manual. https://www.gurobi.com
6. Cebecauer M, Buzna L (2018) Large-scale test data set for location problems. Data in brief 17:267–274
7. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. nature 518(7540):529–533
8. Watkins CJCH, Dayan P (1992) Q-learning. Machine learning 8(3-4):279–292
9. Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. In: Advances in neural information processing systems, pp 1008–1014
10. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: International conference on machine learning, PMLR, pp 387–395
11. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. In: International conference on machine learning, PMLR, pp 1889–1897
12. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv:1509.02971
13. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:1707.06347
14. Babaeizadeh M, Frosio I, Tyree S, Clemons J, Kautz J (2016) Reinforcement learning through asynchronous advantage actor-critic on a gpu. arXiv:1611.06256
15. Levine S, Finn C, Darrell T, Abbeel P (2016) End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research 17(1):1334–1373
16. Deng Y, Bao F, Kong Y, Ren Z, Dai Q (2016) Deep direct reinforcement learning for financial signal representation and trading. IEEE transactions on neural networks and learning systems 28(3):653–664
17. Zheng G, Zhang F, Zheng Z, Xiang Y, Yuan NJ, Xie X, Li Z (2018) Drn: A deep reinforcement learning framework for news recommendation. In: Proceedings of the 2018 World Wide Web Conference, pp 167–176
18. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. nature 529(7587):484–489
19. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T et al (2018) A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 362(6419):1140–1144
20. Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T et al (2020) Mastering atari, go, chess and shogi by planning with a learned model. Nature 588(7839):604–609
21. Jin C, Allen-Zhu Z, Bubeck S, Jordan MI (2018) Is q-learning provably efficient?. arXiv:1807.03765
22. Jin C, Liu Q, Miryoosefi S (2021) Bellman eluder dimension: New rich classes of rl problems, and sample-efficient algorithms. arXiv:2102.00815
23. Duan Y, Jin C, Li Z (2021) Risk bounds and rademacher complexity in batch reinforcement learning. arXiv:2103.13883
24. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) Deep reinforcement learning: A brief survey. IEEE Signal Proc Mag 34(6):26–38
25. Mousavi SS, Schukat M, Howley E (2016) Deep reinforcement learning: an overview. In: Proceedings of SAI Intelligent Systems Conference, Springer, pp 426–440
26. Nguyen TT, Nguyen ND, Nahavandi S (2020) Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. IEEE transactions on cybernetics 50(9):3826–3839
27. Asim M, Wang Y, Wang K, Huang P-Q (2020) A review on computational intelligence techniques in cloud and edge computing. IEEE Transactions on Emerging Topics in Computational Intelligence 4(6):742–763
28. Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. In: Advances in neural information processing systems, pp 2692–2700
29. Lu H, Zhang X, Yang S (2019) A learning-based iterative method for solving vehicle routing problems. In: International Conference on Learning Representations
30. Manchanda S, Mittal A, Dhawan A, Medya S, Ranu S, Singh A (2019) Learning heuristics over large graphs via deep reinforcement learning. arXiv:1903.03332
31. Mazyavkina N, Sviridov S, Ivanov S, Burnaev E (2021) Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research, p 105400
32. Cappart Q, Chételat D, Khalil E, Lodi A, Morris C, Veličković P (2021) Combinatorial optimization and reasoning with graph neural networks. arXiv:2102.09544
33. Nowak A, Villar S, Bandeira AS, Bruna J (2017) A note on learning algorithms for quadratic assignment with graph neural networks. stat 1050:22
34. Kool W, Van Hoof H, Welling M (2019) Attention, learn to solve routing problems!. 7th International Conference on Learning Representations, ICLR 2019, pp 1–25. 1803.08475
35. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2017) Graph attention networks. arXiv:1710.10903
36. Wu Y, Song W, Cao Z, Zhang J, Lim A (2021) Learning improvement heuristics for solving routing problems. IEEE Transactions on Neural Networks and Learning Systems
37. Fu Z-H, Qiu K-B, Zha H (2020) Generalize a small pre-trained model to arbitrarily large tsp instances. arXiv:2012.10658
38. Kool W, van Hoof H, Gromicho J, Welling M (2021) Deep policy dynamic programming for vehicle routing problems. arXiv:2102.11756
39. Lodi A, Mossina L, Rachelson E (2020) Learning to handle parameter perturbations in combinatorial optimization: an application to facility location. EURO Journal on Transportation and Logistics 9(4):100023

40. Gamrath G, Anderson D, Bestuzheva K, Chen W-K, Eifler L, Gasse M, Gemander P, Gleixner A, Gottwald L, Halbig K et al (2020) The scip optimization suite 7.0
41. Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur J Oper Res 290(2):405–421
42. Vesselinova N, Steinert R, Perez-Ramirez DF, Boman M (2020) Learning combinatorial optimization on graphs: A survey with applications to networking. IEEE Access 8:120388–120416
43. Peng Y, Choi B, Xu J (2021) Graph learning for combinatorial optimization: A survey of state-of-the-art. Data Science and Engineering 6(2):119–141
44. Shazeer N, Lan Z, Cheng Y, Ding N, Hou L (2020) Talking-heads attention. arXiv:2003.02436
45. Joshi CK, Cappart Q, Rousseau L-M, Laurent T, Bresson X (2020) Learning tsp requires rethinking generalization. arXiv:2006.07054
46. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980
47. Laurens, Maaten VD, Geoffrey H (2008) Visualizing data using t-sne. J Mach Learn Res 9(2605):2579–2605
48. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167

Chenguang Wang obtained his M.S. degree in operational research and cybernetics from University of Chinese Academy of Sciences in 2022. His research interests include reinforcement learning and combinatorial optimization.

Congying Han received the P.H. degree in applied mathematics from University of Shanghai Jiaotong University in 2008. Now, she is a professor and doctorate tutor of University of Chinese Academy of Sciences. Her current research interests include optimization model and algorithm, machine learning, and pattern recognition.

Tiande Guo is a professor and doctorate tutor of University of Chinese Academy of Sciences. His current research interests include image processing, machine learning and optimization method & application. He has published some articles such as the journal of PAMI, Image Processing and so on.

Man Ding obtained her M.S. degree in operational research and cybernetics from University of Chinese Academy of Sciences in 2021. Her research interests include machine learning, optimization model and algorithm and combinatorial optimization.