Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach

Lu Duan[1], Yang Zhan[1], Haoyuan Hu[1*], Yu Gong[2], Jiangwen Wei[1], Xiaodong Zhang[1], Yinghui Xu[1]
[1]Zhejiang Cainiao Supply Chain Management Co. Ltd [2]Alibaba Group
{duanlu.dl,zhanyang.zy,haoyuan.huhy,jiangwen.wjw,yuhui.zxd,renji.xyh}@cainiao.com,gongyu.gy@alibaba-inc.com

Abstract

Vehicle routing is an important task of the modern transportation service provider. Optimizing vehicle routes not only can significantly reduce operational costs but also increase customer satisfaction. However, the vehicle routing problem is known as NP-hard and it is still challenging to efficiently solve the large-size problems in real delivery systems. In this paper, rather than designing heuristics, we propose a novel deep learning model to learn a heuristic-like policy that generates the vehicle routing schedule.

Our model is based on the graph convolutional network (GCN) with node feature (coordination and demand) and edge feature (the real distance between nodes) as input and embedded. Separate decoders are proposed to decode the representations of these two features. The output of one decoder is the supervision of the other decoder. We propose a strategy that combines the reinforcement learning manner with the supervised learning manner to train the model. Through comprehensive experiments on real-world data, we show that 1) the edge feature is important to be explicitly considered in the model; 2) the joint learning strategy can accelerate the convergence of the training and improve the solution quality; 3) our model significantly outperforms several well-known algorithms in the literature, especially when the problem size is large; 3) our method is generalized beyond the size of problem instances they were trained on.

CCS Concepts

• **Mathematics of computing** → **Combinatorial optimization**; • **Theory of computation** → **Sequential decision making**; • **Computing methodologies** → **Supervised learning by classification**; • **Applied computing** → *Transportation.*

Keywords

Practical Vehicle Routing Problem; Graph Convolutional Network, Separate Decoders; Reinforcement & Supervised Learning; Joint Learning

* Haoyuan Hu is the corresponding author.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403356

ACM Reference Format:
Lu Duan[1], Yang Zhan[1], Haoyuan Hu[1*], Yu Gong[2], Jiangwen Wei[1], Xiaodong Zhang[1], Yinghui Xu[1] . 2020. Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403356

1 Introduction

The vehicle routing problem (VRP) [30] is a vital scheduling task for many delivery service providers, such as E-commerce warehouse, online-to-offline service providers, and last-mile delivery companies. For example, in the E-commerce warehouse, given the orders of a day, it has to dispatch a fleet of vehicles to deliver the ordered package to the customers. Each vehicle must start and return to the warehouse, and each customer is serviced once by one vehicle (as shown in Figure 1). The total of the fixed cost of vehicles and the travel cost due to the consumption of gas is often a concern for the service providers. Therefore, it is an objective to minimize the cost while planning the vehicle routes.

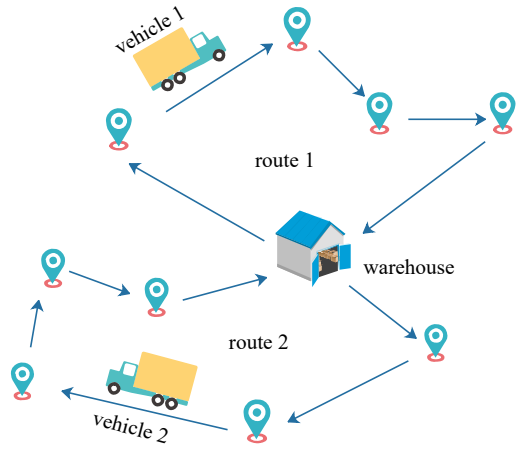


Figure 1: An example of the solution of a VRP problem

The VRP can be summarized as a combinatorial optimization problem that asks "What is the optimal set of routes for a fleet of capacitated vehicles to travel in order to deliver to a given set of customers?". In the simplest form (as the example illustrates), known as the capacitated vehicle routing problem (CVRP), each vehicle is responsible for delivering items to multiple customer nodes, and the total weight of items carried by a vehicle can not exceed its capacity of load; all vehicle must begin and end at a given

node, called the depot; the objective of the problem is to minimize the cost related to the vehicles and travel distance.

However, the VRP is computationally difficult to solve to optimality, even with only a few hundred customer nodes and is classified as NP-hard problem [7]. There are two streams of researches in the literature for solving the VRP. One is based on the techniques of operations research (OR) [22–24] and another is based on deep learning (DL) [6, 21]. The OR-based techniques, such as mathematics optimization, often formulate this problem as a mathematical model [28] and can arrive at (near-) optimal solution by using (heuristic) exact algorithms. However, for practical-size problems, even with only a few hundred customer nodes, these techniques will take several days or even years to find optimal or near-optimal solutions. This makes them hard to implement in practice.

Compared to the OR-based algorithms, the trained policies based on the DL models [21, 26] can solve large-size problems efficiently (often in a few seconds). However, these models are just tested on manually generated problem sets that ignore the complex transportation network in reality. In our computational experiment on solving practical VRP instances, none of their solutions is competitive to a well-known heuristic solver (OR-Tools [10])) that is based on OR techniques. This motivates us to propose a new model that can also outperform the OR-based solver in solution quality.

In this paper, we propose a model based on the graph convolutional networks (GCN) with node and edge features as input and embedded. We propose two separate decoders to decode the embeddings of them. A sequential prediction decoder, based on the recurrent neural networks (RNN), is proposed to output a sequence as the solution for a VRP instance, with the node embedding as input. A classification decoder, based on multi-layer perceptron, is proposed to output a probability matrix, with the edge embedding as input. Specifically, the values in the probability matrix denote the probabilities of edges being present for vehicle routes, which can also be converted to the solution for a VRP instance. We use the output of the sequential prediction decoder as the label (supervision) for the output of the classification decoder, which creates a mutual promotion while learning the two decoders. A reinforcement manner with rollout baseline is used to train the sequential prediction decoder, and a supervised manner with policy-sampling is used to train the other decoder. To train the whole model, a strategy which combines reinforcement and supervised learning manners is used. The trained policy can be viewed as a black-box heuristic (or a meta-algorithm) which generates a high-quality solution in a reasonable amount of time.

We test the method is on different problem sets that are generated based on the data from a real delivery system and compare it with some well-known existing methods. It is shown that our model improves around 5% over other existing learning models [21, 26]. More importantly, it outperforms OR-Tools. Moreover, as the size of the problem increases, the improvements increase, which demonstrates that our method is more capable of solving large-size problems.

To summarize, the contributions of this paper are threefold:

- We present a heuristic-like policy learned by a GCN-based model to solve the practical VRP. A joint learning strategy is proposed to train the model, and it is shown to be effective in accelerating the convergence of the training and improving the solution quality.
- We conduct computational experiments on real-world data sets with real geographic information to validate our method. It is demonstrated that our method outperforms several existing methods. Moreover, our model is shown to be generalized beyond the size of problem instances (the number of nodes in the VRP) they were trained on, which means that we do not need to train the model for each problem size and the learned policy can be used to solve problem instances of any size. This makes our method very appealing in practical application.
- Several insights are obtained for modeling the practical VRP: the GCN is effective to model the VRP graph and it is important to incorporate the edge feature into the network while solving practical VRP.

2 Literature Review

In the literature of operations research, many exact and heuristic algorithms have been proposed to solve VRP. For a comprehensive review, please see [9, 22, 23]. In the following of this section, we focus on reviewing the models or methods in the literature of machine learning, since they are more related to our work.

The first learning algorithm that successfully solve the routing problem can be traced back to [33], which is motivated by neural machine translation [2, 34]. Note that we use *routing problem* to refer to the set of problems that plan routes to deliver to a number of customers, which includes VRP, traveling salesman problem (TSP). The TSP [8], which has only one vehicle and no assignment decision, is a special case of VRP and is computationally easier to solve than VRP. As a result, people usually start with solving TSP to solve other routing problems. [33] introduces the Pointer Network (PN) with attention, to output a permutation of the input. The model is trained off-line supervised by example solutions. It is shown to be able to solve the TSP with customers less than 50 to near-optimal. [3] extends this work by introducing an Actor-Critic algorithm [20] to train the PN without supervised solutions. They consider each instance as a training sample and use the cost (tour length) of a sampled solution for an unbiased Monte-Carlo estimate of the policy gradient. Computational experiments show that they get close to the results by [33] for small instances (20 customers). Moreover, they improve for problem with 50 customers and additionally include results for problem with 50 customers.

Some other learning frameworks are also proposed to solve the TSP. [19] uses a graph embedding structure [5] and a deep Q-learning algorithm [25]. It shows that for a set of randomly generated small-size TSP problem instances, their model slightly worse than a well-known heuristic (farthest insertion), but for some real-world datasets, it slightly outperforms the other. [17, 27] train a Graph Convolution Network (GCN) in a supervised manner to directly output a tour as an adjacency matrix, which is converted into a feasible solution by beam search. [6] replaces the Long Short-Term Memory (LSTM) [15] architecture of [3] by the Transformer architecture [31] and achieve a more efficient learning method. Their framework alone is shown to be as good as OR-Tools (Google's operations research tools [10]) and achieve close to optimal results

when equipped with a simple 2-opt procedure. [18] study a generalization of the TSP, i.e., multiple TSP (mTSP). Given a set of cities, $m \geq 1$ salesmen, the objective of the mTSP is to determine a route for each salesman, such that each city is visited exactly once by any of the salesmen, and the total length of the routes is minimized. The mTSP does not have a capacity constraint for each salesman, which makes it different from the VRP. In the paper, a model based on the Transformer architecture is trained to output a fractional adjacency matrix and then the beam search algorithm is applied to find the best feasible solution to the mTSP.

However, all the models mentioned above do not directly apply to solve VRP since a particular node (e.g. the depot) might be visited multiple times in VRP. Recently, some learning models that are capable of solving VRP were proposed. [26] develops a policy model that uses a recurrent neural network (RNN) decoder with attention to solve VRP. Their experiments show that it performs significantly better than some well-known classical heuristics designed for the VRP, and improves significantly than the OR-Tools VRP engine. [21] develops a framework based on RL to optimize a policy modeled by attention to solve multiple routing problems. Their framework is similar to [6], but they use a different context for the decoder and a more effective training algorithm (using rollout as baseline rather critic). [16] develops a model based on a GCN encoder and RNN decoder to solve the online VRP, where delivery requests are not given but arrive dynamically. They propose a deep reinforcement learning mechanism with an unsupervised auxiliary network to train the model parameters.

However, the models of [6] and [21] are tested on manually generated problem sets that ignore the complex transportation network in reality. In our computational experiment on solving practical VRP instance, none of their results is competitive to a well-known heuristic solver (OR-Tools [10])) that is based on OR techniques. This motivates us to propose a new DL model that has the capability of solving practical VRP instances efficiently and effectively in this paper.

We summarize the difference and similarities between our model and the related methods in Table 1.

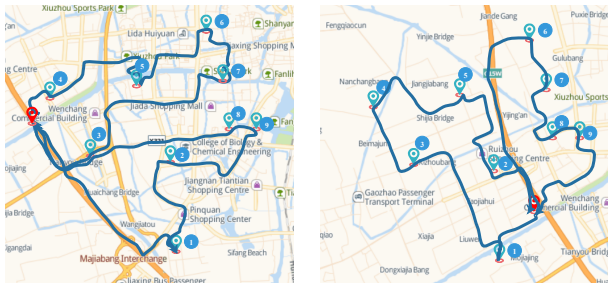
Table 1: Comparison between our model and the related ones

Literature	Problem	Encoder	Embedded features*	Learning manner
[33]	TSP	PN	node	supervised
[3]	TSP	PN	node	reinforcement
[19]	TSP	GNN	node	reinforcement
[27]	TSP	GCN	node	supervised
[17]	TSP	GCN	node & distance matrix	supervised
[6]	TSP	RNN	node	reinforcement
[18]	mTSP	RNN	node	supervised
[26]	VRP	RNN	node	reinforcement
[21]	VRP	RNN	node	reinforcement
[16]	online VRP	GNN	node	reinforcement
Our paper	VRP	GCN	node & distance matrix	reinforcement & supervised

* **Node feature for the TSP is the coordination of each node, while the the VRP it also include the demand.**

Basically, there are two major differences between our models and the others. First, different from most of them using an RNN encoder, our model use a GCN one. Naturally, the VRP can be defined as a Graph with nodes and edges, so graph embedding is a quite straightforward option. Moreover, since we want to incorporate multiple features of the VRP in the model, GCN has a large space to manipulate.

Second, different from other models that also use graph embedding (STRUCT2VEC [5]), we input not only node feature but also edge feature (the distance matrix between nodes) into the graph network. We notice that most of the existing learning models consider solving problems with the assumption that the distance between nodes is Euclidean, which depends on coordination. So they just input the node feature in the network. We consider that while solving the practical vehicle routing problem, it is important to incorporate the real distance between nodes. Because in reality, the travel distance is not only related to the coordination, but also to many complicated factors, such as road network, traffic conditions, and weather. We show a case in Figure 2, where the Euclidean distance matrix of customer nodes are the same in (a) and (b), but the actual travel distance in practice are totally different (note that we obtain the travel distance between two nodes from GIS platform, which is the shortest route). Therefore, in this paper, to learn more information about the real transportation network and enable our trained policy to solve new problems, even with different sizes, we input both node feature (coordination and demand) and edge feature (real travel distance obtained from GIS) in the GCN.



(a) area 1 (b) area 2

Figure 2: An illustration: the real travel distance matrix of customer nodes with the same relative coordination can be different and thus the optimal routes over them may be different.

Although some models consider the edge feature, the effectiveness of is not explicitly shown [17]. [26] indicate that the edge feature did not improve solution quality for solving the VRP. Whereas in this paper, we show that by using a joint learning strategy (combining reinforcement learning with supervised learning), the edge feature can actually improve the results compared with when it is not incorporated.

3 A Novel Joint Learning Approach

In this section, we first give a formal description about the VRP in a graph optimization perspective. We then present the proposed model, which is based on a graph convolutional networks with node sequential prediction and edge classification (GCN-NPEC). It consists of three components, i.e., problem graph embedding encoder, node sequential decoder and edge classification decoder, as present in Section 3.2.

3.1 Problem Setup: Graph Optimization Perspective

A VRP instance can be defined on a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{0, \ldots, n\}$, where node $i = 0$ is the depot and $i \in \{1, \ldots, n\}$ are the customers. Moreover, $\mathcal{E} = \{e_{ij}\}, i, j \in \mathcal{V}$ is the set of edges between nodes in $\mathcal{V}$. The depot node is associated with the coordination, x_0^c, while each customer node i is associated with a two dimensional-feature vector, $x_i = \{x_i^c, x_i^d\}$, where x_i^c is the coordination and x_i^d is the demand. Each edge is associated with the distance between the connected nodes, m_{ij}. Every two nodes are connected, but due to the complexity of the traffic network in reality, we consider that the distance from node i to j (m_{ij}) is different from j to i (m_{ji}). In other words, the graph is directed full-connected. This also indicates that edge e_{ij} is different from edge e_{ji}.

Given that: (i) the VRP is to find a set of tours, where each tour corresponds to the routing of a vehicle and it begins and ends at node 0; (ii) each customer is traveled exactly once, and the carrying weight of each vehicle can not exceed its capacity; (iii) the objective is to minimize the total cost, which usually includes the fixed cost of vehicle and traveling cost. In our model, we aim to produce a sequence of customers $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \ldots, \pi_T)$, where $\pi_t \in \{0, 1, 2, \ldots, n\}$ and 0 may incur multiple times but other nodes incur just once. Therefore, the sequence between every two 0s is the routing for a vehicle. The objective of our model can be denoted as:

$$\min \; c_v Q_v + c_t \sum_{t=1}^{T-1} m_{\pi_t \pi_{t+1}} \tag{1}$$

where c_v is the fixed cost of a vehicle, Q_v is the number of used vehicles, and c_t is the unit cost of traveling.

3.2 Graph Convolutional Networks with Node Sequential Prediction and Edge Classification

The GCN-NPEC model follows the general encoder-decoder perspective. Figure 3 depicts the overall architecture of it. The encoder produces representations of the nodes and edges in graph $\mathbb{G}$. Different from most of the existing models, we have two decoders in the model, and we call them sequential prediction decoder and edge classification decoder. The sequential prediction decoder takes the node representations as input and produces a sequence $\boldsymbol{\pi}$ as the solution for a VRP instance. While the classification decoder is used to output a probability matrix that denotes the probabilities of the edges being present for vehicle routes, which can also be converted to the solution for a VRP instance. In this way, we use $\boldsymbol{\pi}$ as the ground-truth (label) of the output of the classification decoder. While training the model, we use a joint strategy, which combines reinforcement and supervised learning manners.

3.2.1 Input Given the graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ of a VRP instance, the input representation for each node $i \in \mathcal{V} \backslash \{0\}$ is the associated of the demand and the coordination, and that of the depot 0 is associated with the coordination. Specifically, they are initialed as d_x-dimensional vectors by using a simple fully connected neural network with nonlinear activation as follows:

$$x_i = \begin{cases} Relu(W_1 x_{c_0} + b_1), & if\ i = 0, \\ Relu([W_2 x_i^c + b_1; W_3 x_i^d + b_2]), & if\ i \geq 0. \end{cases} \tag{2}$$

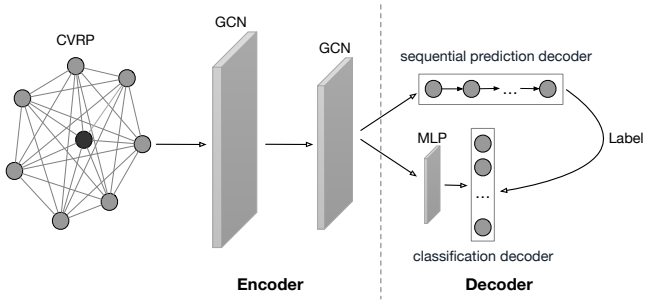


Figure 3: The overall architecture of our GCN-NPEC model.

where $[;]$ represents the concatenation operation, and W_1, W_2, W_3, b_1, b_2 , b_3 are trainable parameters.

For each edge e_{ij} in $\mathcal{E}$, the edge feature y_{ij} is associated by its distance and the adjacency between nodes (i, j). Although $\mathbb{G}$ is fully connected, in fact, we usually define the adjacency matrix $A \in R^{(n+1)\times(n+1)}$ as follows:

$$a_{ij} = \begin{cases} 1, \ , if\ i\ and\ j\ are\ k - nearest\ neighbors, \\ -1, \ if\ i = j, \\ 0, \ others. \end{cases} \tag{3}$$

Thus, if node i and node j are k-nearest neighbors (we use $k = 10$ in this paper), they are adjacent; otherwise they are not. Moreover, we use -1 to indicate self-connection. Then the edge feature y_{ij} is initialed as a d_y-dimensional vector :

$$y_{ij} = Relu([W_4 m_{ij} + b_4; W_5 a_{ij} + b_5]) \tag{4}$$

where W_4, W_5, b_4 and b_5 are trainable parameters.

3.2.2 GCN Encoder Given the d_x-dimensional node feature x_i and d_y-dimensional edge feature y_{ij}, the encoder firstly computes th initial d_h-dimensional graph node and edge embeddings (a.k.a representations), i.e., h_i^0 and $h_{e_{ij}}^0$, respectively, by using the learned linear projection with parameters W_{E1}, W_{E2}, b_{E1} and b_{E2}:

$$h_i^0 = W_{E1} x_i + b_{E1} \tag{5}$$

$$h_{e_{ij}}^0 = W_{E2} y_{ij} + b_{E2}. \tag{6}$$

The embeddings are then updated through L graph convolutional (GC) layers, each consisting of two sub-layers: aggregation and combination [35, 36].

In a standard GCN with L layers, each layer ℓ ($\ell = 1, 2, \ldots, L$) has an aggregation and a combination sub-layer, through which the node embedding at ℓ-th layer is defined by:

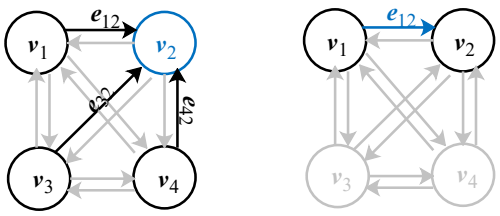
$$h_{N(i)}^\ell = \sigma\left(W^\ell AGG\left(\{h_{i'}^{\ell-1}, \forall i' \in N(i)\}\right)\right) \tag{7}$$

$$h_i^\ell = COMBINE(h_i^{\ell-1}, h_{N(i)}^\ell) \tag{8}$$

where $h_{N(i)}^\ell$ denotes the aggregated feature of node i's neighborhood, $N(i)$ is a set of nodes adjacent to i, and AGG is a function used for aggregating embeddings from the neighbors of node i, which can be customized for specific models, such as max-pooling, mean-pooling [12] or attention based weighted summation [32]. W^ℓ is a trainable matrix shared among all nodes at layer ℓ. σ is a non-linear activation function, e.g., Relu. The $COMBINE$ function is used to combine the self embedding and the aggregated embeddings, which is also a custom setup for different graph models, e.g., concatenation as in GraphSAGE [12].

In this paper, considering the features of the VRP graph, we tailor the standard GCN. Different from the standard GCN that treats all kinds of nodes as the same and do not input the edge

feature, our GCN input both the node and edge features and update them simultaneously. A simplified illustration of the updates of embeddings in a GC layer is shown in Figure 4. The details of our aggregation and combination sub-layers are defined as follows.



(a) Node Update (b) Edge Update

Figure 4: Illustration of updates in a GC layer. The blue indicates the element that is being updated, and the black indicates other elements which are involved in the update (note that the pre-updated value of the blue element is also used in the update).

Aggregation sub-layer. For a node $v_i \in \mathcal{V}$, the aggregated embedding from neighbors, $h^{\ell}_{N(v)}$ is updated as:

$$h^{\ell}_{N(i)} = \sigma\left(W^{\ell}_I AGG^{\ell}_I(ATTN(h^{\ell-1}_i, \{h^{\ell-1}_{v'}, \forall v' \in N(i)\})\right) \tag{9}$$

where W^{ℓ}_I is a trainable parameter and $ATTN$ is a function $f : h_{key} \times H_{val} \rightarrow h_{val}$ which maps a feature vector h_{key} and the set of candidates' feature vectors H_{val} to an weighted sum of elements in H_{val}. The weights of the summation, i.e. attention values are calculated by the scaled dot-product attention [31], which makes the diffusion process anisotropic on graphs.

For each edge $e_{ij} \in \mathcal{E}$, we consider that the nodes it links to as its neighbors. Thus, the aggregated embedding associated with the edge, $h^{\ell}_{N(e_{ij})}$, is defined as:

$$h^{\ell}_{N(e_{ij})} = \sigma\left(W^{\ell}_E AGG^{\ell}_E\left(\{h^{\ell-1}_{e_{ij}}, h^{\ell-1}_i, h^{\ell-1}_j\}\right)\right) \tag{10}$$

$$AGG^{\ell}_E\left(\{h^{\ell-1}_{e_{ij}}, h^{\ell-1}_i, h^{\ell-1}_j\}\right) = W^{\ell}_{e1} h^{\ell}_{e_{ij}} + W^{\ell}_{e2} h^{\ell}_i + W^{\ell}_{e3} h^{\ell}_j \tag{11}$$

where W^{ℓ}_E, W^{ℓ}_{e1}, W^{ℓ}_{e2} and W^{ℓ}_{e3} are trainable parameters.

Combination sub-layer. Given the aggregated embeddings, we define the combination layer by following the strategy in [12] as:

$$h^{\ell}_i = \left[V^{\ell}_I h^{\ell-1}_i; h^{\ell}_{N(i)}\right] \tag{12}$$

$$h^{\ell}_{e_{ij}} = \left[V^{\ell}_E h^{\ell-1}_{e_{ij}}; h^{\ell}_{N(e_{ij})}\right], \tag{13}$$

where V^{ℓ}_I and V^{ℓ}_E denote trainable weight matrices for node and edge, the h^{ℓ}_i and $h^{\ell}_{e_{ij}}$ are the node and edge hidden state of in the ℓ-th layer. We emphasize that those trainable parameters mentioned above are unique per layer. Furthermore, each sub-layer adds a skip-connection [13] and layer normalization [1] operations.

3.2.3 The Decoders Given the node and edge embeddings produced by the GCN encoder, we propose two networks to separately decode them:

Sequential prediction decoder. Following the previous literature, we use a recurrent neural networks with a Gated recurrent unit (GRU) [4] and a context-based attention mechanism to map the nodes embedding to an sequence $\boldsymbol{\pi}$. The reason we use a GRU cell rather than the self-attention mechanisms used in [21] is that the sequential solution is heavily based on previous steps.

Given an arbitrary input S, this process will generate a sequence of length T, $\boldsymbol{\pi} = \{\pi_t, t = 1, \ldots, T\}$, possibly with a sequence length larger than $n + 1$ because node 0 may incur multiple times. We also use the notation $\boldsymbol{\pi}_t$ to denote the decoded sequence up to time t, i.e., $\boldsymbol{\pi}_t = \pi_1, \ldots, \pi_t$. We are interested in finding a stochastic policy $p(\boldsymbol{\pi}|S;\theta)$ which generates the sequence $\boldsymbol{\pi}$ in a way that minimizes the objective as shown in Equation (1) while satisfying the problem constraints. The stochastic policy is the joint probability and can be decomposed by the chain rule as follows

$$\begin{aligned} P(\boldsymbol{\pi}|s;\theta) &= \prod_{t=0}^{T} p(\pi_{t+1}|S, \boldsymbol{\pi}_t;\theta) \\ &= \prod_{t=0}^{T} p(\pi_{t+1}|f(S,\theta_e), \boldsymbol{\pi}_t;\theta_d), \end{aligned} \tag{14}$$

where $f(S;\theta_e)$ is the encoder and θ_d is a trainable parameter. Here we use a GRU cell to estimate the last term in the above Equation (14) by introducing a state vector, z_t, which embeds the already generated outputs $\boldsymbol{\pi}_{t-1}$, i.e.

$$p(\pi_t|f(S,\theta_e), \boldsymbol{\pi}_{t-1};\theta_d) = p(\pi_t|f(S,\theta_e), z_t;\theta_d). \tag{15}$$

Decoding happens sequentially, and at decoding step $t \in \{1, \ldots, T\}$, the sequential decoder outputs the node π_t based on the node embedding from the GCN encoder and the GRU hidden state z_t. During decoding, $p(\pi_t|z_t, f(S,u;\theta_e);\theta_d)$ is implemented by a specific attention mechanism named **pointer** [33], in which it will attend to each node in the encoded graph and calculate the attention scores before applying softmax function to get the probability distribution. It allows the decoder to look at the whole input graph $\mathbb{G}(\mathcal{V},\mathcal{E})$ at any time and select a member of the input nodes $\mathcal{V}$ as the final outputs $\boldsymbol{\pi}$.

For notation purpose, let h^L_i be the embedded input, at decoding step t, the context weight of node i, denoted by u_{ti}, is computed as

$$u_{ti} = \begin{cases} -\infty, & \forall j \in N_{mt} \\ h^T_a \tanh(W^G\,[v_i; z_t]), & \text{otherwise,} \end{cases} \tag{16}$$

where N_{mt} is the set of masked nodes at step t, and $h^T_a \in \mathbb{R}^{d_h}$, $W^G \in \mathbb{R}^{1\times 2}$ are parameters. Note that before computing the context weight vector, we have a masking procedure that masks the nodes that are infeasible to be decoded at step t. The masking rules depend on the feasibility constraints of the problem.

In the VRP, each vehicle (route) has capacity $c > 0$ and each node has a demand, denoted by x^d_i. We have $0 < x^d_i < c, \forall\, i \in \{1, 2, \ldots, N\}$ and $x^d_0 = 0$. There is a feasibility constraint for a solution is that the total demand in each route should not exceed the capacity, i.e., $\sum_{i\in R_j} x^d_i \leq c$, where R_j is the set of node indexes assigned to route j.

To facilitate the capacity constraints, we keep track of the remaining demands $\tilde{x}^d_i$ for the customers nodes $i \in \{1, \ldots, N\}$ and the remaining vehicle capacity $\tilde{c}_t$ at time t. If a node is visited, we set $\tilde{x}^d_i = 0$. The remaining capacity is initialized and update as

follows:

$$\tilde{c}_t = \begin{cases} c, & \pi_t = 0. \\ \max(0, \tilde{c}_{t-1} - x^d_{\pi_t}), & \pi_t \neq 0. \end{cases} \tag{17}$$

At step t, we do not allow a node to be visited if its remaining demand exceed the remaining capacity or if the node was already visited. Moreover, we do not allow the depot to be visited at time $t = 1$ or at two subsequent time steps, although it can be visited multiple times. Therefore, the set of masked nodes at step t is :

$$N_{mt} = \begin{cases} N_{m(t-1)} \bigcup \{0\} & \pi_{t-1} = 0 \; or \; t = 1 \\ \bigcup \{i | \tilde{x}^d_i = 0 \; or \; \tilde{x}^d_i \geq \tilde{c}_t\}, & others \end{cases} \tag{18}$$

Then softmax function is applied to get the pointing distribution towards input nodes, as follows

$$p(\pi_t | f(S, \theta_e), \boldsymbol{\pi}_{t-1}; \theta_d) = \text{softmax}(u_{ti}), \; j \in \{1, ..., N\} \tag{19}$$

Classification decoder. To use the graph edge embedding to generate a solution, previous works [17] usually use the optimal solution as supervision (label) to train the decoder. However, this method does not work in our case, because our aim is to solve large-size practical VRP, whose optimal solution are rarely given nor can be easily obtained by using other methods.

This motivates us to develop a new method. Intuitively, since both node embeddings and edge embeddings own the graph information and intact with each other, the solution generated from the sequential decoder should be the same as that generated from the classifier decoder, when the method converges. Therefore, we can use the solution $\boldsymbol{\pi}$ produced by the sequential prediction decoder as a supervised label for the classifier decoder. Note that a sequence $\boldsymbol{\pi}$ can be converted into one and exactly one set of present (selected) edges as vehicle routes. For example, sequence $\{0, 4, 5, 1, 0, 2, 3, 0\}$ corresponds to that the set of edges $\{e_{04}, e_{45}, e_{51}, e_{10}, e_{02}, e_{23}, e_{30}\}$ are present for the vehicle routes. Either of them can denote the solution for a VRP instance. Let the values of the present edges be 1, and the values of the absent edges be 0, we can obtain a 0-1 matrix corresponding a sequence such that $P^{VRP*}_E = \{p^{\text{VRP}^*}_{e_{ij}}\}_{e_{ij} \in \mathcal{E}}$.

To use P^{VRP*}_E as the label, we use a Multi-layer Perceptron (MLP) over the edge embedding of the last GCN layer, $h^L_{e_{ij}}$, then to obtain a softmax distribution, which can be regarded as the probability of edge e_{ij} being present. Specifically, the value is defined as:

$$p^{\text{VRP}}_{e_{ij}} = \text{softmax}(MLP(h^L_{e_{ij}})) \in [0, 1]^2 \tag{20}$$

In the following, we use P^{VRP}_E to denote the probability matrix obtained by MLP layer. Above all, with edge embedding as input, the proposed classification decoder should output P^{VRP}_E as close as P^{VRP*}_E. This architecture enables the edge classifier to be improved by a well-predicted sequence. Simultaneously, with the information from edge features incorporated, the sequential prediction decoder can be improved. This forms a virtuous circle for the learning of both decoders.

3.2.4 A Joint Learning Strategy The above present model has the framework for reinforcement learning (RL) and supervised learning (SL). To train it, naturally, we use a joint strategy that combines RL and SL.

REINFORCE with rollout baseline To do that, we first define the reinforced loss of the sequential prediction decoder as the expected cost: $\mathcal{L}_r(\theta_e, \theta_d | S) = \mathbb{E}_{P(\boldsymbol{\pi} | f(S;\theta_e);\theta_d)} L(\boldsymbol{\pi})$, where $L(\boldsymbol{\pi})$ the total cost of the solution $\boldsymbol{\pi}$. We use policy gradient based reinforcement learning (REINFORCE) [29] with a baseline $b(S)$ to train the policy. And its loss function is derived as follows:

$$\begin{aligned} \mathcal{L}_r(\theta_e, \theta_d) &= \sum_S \mathbb{E}_{\boldsymbol{\pi} \sim P(\boldsymbol{\pi} | S; \theta_e, \theta_d)} (L(\boldsymbol{\pi}) - b(s)) \\ &= \sum_S (L(\boldsymbol{\pi}) - b(S)) \sum_{i=1}^{T} \log p(\pi_i | \boldsymbol{\pi}_{i-1}, S; \theta_e, \theta_d) \end{aligned} \tag{21}$$

where $b(S)$ is the cost of a solution from a deterministic greedy rollout policy proposed by [21]. The baseline is fixed for each epoch by freezing greedy rollout policy. At the end of every epoch, the parameters of the baseline policy will be updated if the current trained policy improves significantly (according to a paired t-test). Then the baseline will update. In the following, we denote the trained policy by $\mathbb{P}_\theta$, and the baseline policy by $\mathbb{P}_{\theta^{BL}}$.

SUPERVISE with policy-sampling We use this strategy to train the classification decoder. Specifically, given a sequence $\boldsymbol{\pi}$ from the sequential prediction decoder, we convert it into a 0-1 matrix, $P^{\text{VRP}^*}_E$, as introduced above. We define the loss function for supervising as:

$$\mathcal{L}_s(\theta_e, \theta_c) = -\sum_{S, \boldsymbol{\pi}} crossEntropy(P^{VRP}_E, P^{\text{VRP}^*}_E) \tag{22}$$

where θ_c is a trainable parameter. To combine REINFORCE and SUPERVISE, we simply apply linear combination of their loss functions, and conduct the final loss as follows:

$$\mathcal{L}_{\boldsymbol{\theta}} = \alpha \times \mathcal{L}_s(\boldsymbol{\theta}) + \beta \times \mathcal{L}_r(\boldsymbol{\theta}) \tag{23}$$

where α and β are the hyper-parameters to be tuned and $\boldsymbol{\theta}$ denotes the vector of trainable parameters. In our experiments, we set $\alpha = 1.0$ and $\beta = 1.0$ .The overall learning process is shown in Algorithm 1. Firstly, we define a baseline policy $\mathbb{P}_{\theta^{BL}}$ as the trained policy $\mathbb{P}_\theta$. As the model changes during training, the baseline policy $\mathbb{P}_{\theta^{BL}}$ will be frozen for a fixed amount of steps (every epoch) until the improvement is significant as determined by a paired t-test with $\gamma = 5\%$. Line 6 shows the generation of graph node embeddings H_{I_i} and edge embeddings H_{E_i} by a GCN-Encoder. Then we use the sequential prediction decoder to decode the node embeddings H_{I_i} to output two solutions: $\boldsymbol{\pi}_i$, $\boldsymbol{\pi}^{BL}_i$, with policy $\mathbb{P}_\theta$ and $\mathbb{P}_{\theta^{BL}}$, respectively. Line 9 converts the sampled solution $\boldsymbol{\pi}_i$ to a 0-1 matrix as the ground truth for the classification decoder.

4 Computational Experiment

In this computational experiment, we test our method and compare it with other existing methods on a set of real VRP instances, which are generated based on the data from our partner – a large B2B platform in China [1]. One of its warehouse in Hangzhou needs to delivery packages to an amount of stores located throughout the city everyday.

[1] https://8.1688.com

Algorithm 1 The learning procedures of the GCN-NPEC model

Input: number of epochs E, steps per epoch T, batch size B, significance γ (for paired t-test)
Initialization: $\theta, \theta^{BL} \leftarrow \theta$
for epoch = $1, \ldots, E$ **do**
  for step=$1, \ldots, T$ **do**
    $s_i \leftarrow$ RandomInstance() $\forall i \in \{1, \ldots, B\}$
    $H_{I_i}, H_{E_i} \leftarrow$ GCN-Encoder(s_i) $\forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\pi}_i \leftarrow$ SampleRollout$(H_{I_i}, \mathbb{P}_\theta)$ $\forall i \in \{1, \ldots, B\}$;
    $\boldsymbol{\pi}_i^{BL} \leftarrow$ GreedyRollout$(H_{I_i}, \mathbb{P}_{\theta^{BL}})$ $\forall i \in \{1, \ldots, B\}$;
    $P_{E_i}^{VRP^*} \leftarrow \boldsymbol{\pi}_i$
    $P_{E_i}^{VRP} \leftarrow$ ClassificationDec$(H_{E_i}, \mathbb{P}_\theta)$ $\forall i \in \{1, \ldots, B\}$
    $\mathcal{L}_s(\theta) \leftarrow -\sum_{i=1}^{B}$(cross-entropy$(P_{E_i}^{VRP}, P_{E_i}^{VRP^*})$)
    $\mathcal{L}_r(\theta) \leftarrow \sum_{i=1}^{B} \left(L(\boldsymbol{\pi}_i) - L(\boldsymbol{\pi}_i{}^{BL})\right) \log \mathbb{P}_\theta(\boldsymbol{\pi}_i)$;
    $\mathcal{L}_\theta \leftarrow \alpha \times \mathcal{L}_s(\theta) + \beta \times \mathcal{L}_r(\theta)$
    $\theta \leftarrow$ Adam$(\theta, \nabla\mathcal{L})$;
  if OneSidedPairedTTest$(\theta, \theta^{BL}) < \alpha$ **then**
    $\theta^{BL} \leftarrow \theta$;

4.1 Experimental Settings

4.1.1 Generation of Problem Sets We generate six problem sets based on the real data from the delivery system of the B2B platform, whose details are summarized in Table 2. Each problem set includes instances for training and testing. For problem set VRP20, VRP50, VR100, VRP200 and VRP400, which has 20, 50, 100, 200, 400 customers node in each instance, respectively, the locations of customers and the depot are randomly selected from the historical orders from the platform. Moreover, the discrete demand of each customer node is uniformly chosen from $\{1, .., 9\}$. Specifically, for different problem sizes, we consider different vehicle capacities, namely, $c^{20} = 30, c^{50} = 40, c^{100} = 50, c^{200} = 60, c^{400} = 80$, where the subscript is the size of problem.

For problem set VRP-R, each instance is real and from orders made in the platform in Sep. 2019. Specifically, the samples of the first 15 days are used for training and the samples of the following 15 day are for testing. In this set, the problem sizes vary from 20 to 300.

For all the problem sets, the distance between two nodes are provided by GaoDe map api[2].

4.1.2 Methods of Implementation To test the effectiveness of the GCN-NPEC model, we compare it with several existing methods.

The VRP can be formulated as a mixed-integer programming (MIP) model, and solved by using some state-of-the-art commercial solvers (e.g. Gurobi [11]). However, we find out that Gurobi (8.1.1) fails to converge to optimality in two hours, even for problem with 20 customers. Since there are 10,000 instances, it is unrealistic to finish the experiment in reasonable time if we use Gurobi to obtain optimal solutions. Therefore, we give up to compare our method with Gurobi and known optimal solutions.

[2]https://lbs.amap.com/

Table 2: The generated problem sets: the data of instances in VRP-R are from the real orders made in the platform in Sep. 2019 and that in other sets are randomly selected from the historical orders.

Problem set	Problem size	Locations	Demand	Data Size(train/test)
VRP20	20	randomly selected	uniformly chosen	50000 / 10000
VRP50	50			50000 / 10000
VRP100	100			50000 / 10000
VRP200	200			25600 / 5000
VRP400	400			25600 / 5000
VRP-R	20-300	real	real	30000 /10000

As an alternative method, we consider the following well-known methods as our baseline, which has been shown to be able to produce near-optimal solutions on other problem sets:

- OR-Tools: Google's operations research tools [10], tuned for tackling the world's toughest problems in vehicle routing. It is a heuristic solver that can produce solutions for the large-size VRPs in a reasonable amount of time.
- PRL: a learning model proposed in [26], which is based on deep neural network with a LSTM encoder. It is shown to outperform OR-Tools and it does not require an explicit distance matrix.
- AM: a learning model proposed in [21], which is based on the attention model with coordination embeddings and does not consider real distance matrix. It is shown to outperform a well-known heuristic for VRP, LKH3 heuristic [14].

OR-Tools represents the methods based on OR techniques, and PRL and AM are DL-based methods, just like our method.

Moreover, to demonstrate the effectiveness of the edge classification decoder and the joint learning strategy, we train a GCN-NPEC model without the classification decoder (which is referred to as GCN-Node in the following) for comparison.

4.1.3 Hyper-parameters For all learning models, we use mini-batches of fixed size 256 (except for VRP with $n = 200, 400$, where we use 64 for memory constraints). For the GCN-based models, we use 3-layer GCN as encoder, 2-layer GRU networks with 256 hidden units for the sequential prediction decoder and use 3-layer fully-connected networks for classification decoder with hidden units 256, 256, 1, respectively. All the parameters are initialized randomly in $[-0.08, 0.08]$ and clip $L2$ norm of our gradients to 5.0. In addition, the description of each node's and edge's feature is embedded into a 256-dimensional input. We train our model with the Adam optimizer by an initial learning rate of 10^{-3} and decay by a factor of 0.96 per epoch until it reaches to the minimal $3 * 10^{-6}$. We train for 1000 epochs using the training data as described above. Moreover, all models are trained with the rollout baseline policy, and the performances are reported with greedy and beam search strategy at the inference time. Beam search is a popular approach for obtaining a set of high-probability sequences from generative models for natural language processing tasks [34]. As the best quality solution may not be same as the solution of the highest probability, we also report the results of the best quality solutions at the end of beam search.

4.1.4 Evaluation Metric For the instances in problem set VRP20, VRP50, VR100, VRP200 and VRP400, the fixed cost of a vehicle is 0

and the unit cost of travel distance is 1. We calculate the cost for each solution based on these parameters. Since there are 10,000 test instance in each of these sets (5000 instances for VRP200 and VRP400), we report the average cost of each method over each problem set. Moreover, we report the gap between the DL-based method and the OR-Tools, which is calculated by $\frac{\tilde{C}_{ML_i} - \tilde{C}_{OR-Tools}}{\tilde{C}_{OR-Tools}} \times 100\%$, where $\tilde{C}_{ML_i}$ is the average cost of a certain DL-based method and $\tilde{C}_{ML_i}$ is the average cost of OR-Tools.

For the instances in problem set VRP-R, the fixed cost of vehicle is 100 and the unit cost of travel distance is 1. Similarly, we obtain the average cost and gap of each method over the testing instances in the problem set.

4.2 Computational Results

We show the performance of all the methods in Table 3. The table is divided into four sections: the heuristic solver, and the DL-based models with greedy serach (G), beam search methods (BM) and beam search methods with the best quality solutions (BMS), respectively. Note that the costs are normalized by a scale of $1e^5$.

From the table, firstly, we can see that the DL-based models with BMS have the best performance over those with other search methods, which indicates that BMS outperforms other search methods. Secondly, GCN-based models significantly outperform (improves around 5% over) other learning models (based on RNN) under all search methods, which indicates that GCN is very effective for modeling the VRP graph. Thirdly, from the relative improvement values of GCN-NPEC over GCN-Node in most of the cases, we can conclude that the supervision of edge classification decoder is beneficial and the joint learning strategy is effective. Moreover, it demonstrates that the edge feature can improve solution quality, which disproves the conclusion in [26]. More closely, we can see that as the problem size increases, the improvement of GCN-NPEC over other methods increases, which demonstrates that our method is more capable of solving large-size problems. We conjecture the underlying reason is that the complexity of the real transportation network increases in the problem size, and our method learns it more effectively than PRL and AM. Finally, we can see that our method based on the GCN-NEPC model can outperform OR-Tools regarding the solution quality. Regarding the computational time, our trained policy can solve a problem instance in a few seconds, while OR-Tools needs several hours. Therefore, our method can be more effective and efficient than the heuristics based on OR techniques.

We also show the comparison of learning curves in Figure 5. From the curves, we can see that the GCN-based methods converge faster than other DL-based methods, which verifies that the edge feature can bring additional information of the graph to the model and it is important to incorporate it into the network while solving the (practical) VRP. Moreover, GCN-NPEC converges faster than GCN-Node, verifying that the joint learning strategy is effective. We speculate that there is a mutual promotion between the learning of the sequential prediction decoder and the classifier decoder, so the learning could converge faster.

We show the performance of the methods on solving the problem set VRP-R in Table 4. We can see that GCN-NPEC also performs

Table 3: The performance of different methods. The percentage is the solution gap over OR-Tools and the float number is average cost (normalized by a scale of $1e^5$). In Model column, (G): greedy search, (BM): beam search and (BMS): the beam search with highest solution quality.

Model	Problem set				
	VRP20	VRP50	VRP100	VRP200	VRP400
OR-Tools	4.420 (0.00%)	7.493 (0.00%)	11.348 (0.00%)	17.995 (0.00%)	26.095 (0.00%)
PRL(G)	4.731 (7.03%)	7.671 (2.38%)	12.081 (6.56%)	20.021 (11.26%)	30.197 (15.72%)
AM(G)	4.618 (4.47%)	7.501 (0.11%)	11.965 (5.44%)	18.986 (5.11%)	29.589 (13.39%)
GCN-Node(G)	4.481 (1.38%)	7.499 (0.08%)	11.347 (-0.01%)	17.894 (-0.56%)	27.808 (6.56%)
GCN-NPEC(G)	4.480 (1.36%)	7.497 (0.05%)	11.346 (-0.02%)	17.819 (-0.09%)	27.428 (5.11%)
PRL(BM)	4.542 (2.76%)	7.532 (0.52%)	11.907 (4.93%)	19.521 (8.48%)	28.846 (10.54%)
AM(BM)	4.521 (2.28%)	7.422 (-0.95%)	11.746 (3.51%)	18.697 (3.90%)	27.143 (4.02%)
GCN-Node(BM)	4.402 (-0.41%)	7.312 (-2.42%)	11.276 (-0.06%)	17.679 (-1.75%)	27.037 (-3.60%)
GCN-NPEC(BM)	4.401 (-0.42%)	7.306 (-2.50%)	11.266 (-0.07%)	17.589 (-2.25%)	26.977 (3.38%)
PRL(BMS)	4.484 (1.44%)	7.530 (0.50%)	11.901 (4.87%)	19.019 (5.69%)	28.438 (8.98%)
AM(BMS)	4.415 (0.11%)	7.421 (-0.96%)	11.643 (2.60%)	18.516 (2.90%)	26.781 (2.63%)
GCN-Node(BMS)	4.399 (-0.46%)	7.312 (-2.42%)	11.259 (-0.08%)	17.660 (-1.86%)	26.093 (-0.01%)
GCN-NPEC(BMS)	**4.398 (-0.50%)**	**7.304 (-2.52%)**	**11.186 (-1.43%)**	**17.516 (-2.66%)**	**25.876 (-0.84%)**

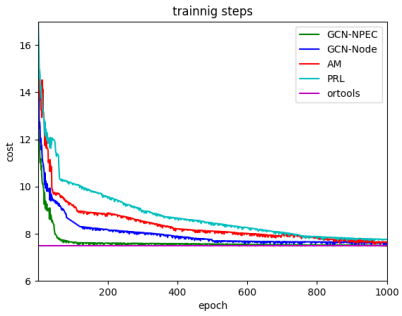


Figure 5: learning curves for VRP50

the best, which illustrates that our model is generalized beyond the size of problem instances they were trained on, which means that we do not need to train the model for each problem size and the learned policy can be used to solve problem instances of any size. This makes our model very appealing in practical application. .

Table 4: Average cost for different methods on problem set VRP-R. The values are normalized by a scale of $1e^5$.

Model	Problem set
	VRP-R
OR-Tools	131.596
PRL(BMS)	139.438
AM(BMS)	135.741
GCN-Node(BMS)	130.634
GCN-NPEC(BMS)	**129.949**

Moreover, some example solutions obtained by using GCN-NPEC can be seen in Figure 6 in Appendix A . These visualizations give some insights of the heuristic-like policy learned by the model. Interestingly, we can see that there are some visually unreasonable paths such as circles, which may due to the complexity of the real transportation network.

5 Conclusion

Vehicle routing is an important task of the modern transportation service provider. It is a computationally difficult problem and is classified as NP-hard. Specifically, for large-size problems with real geographic information considered, it is very challenging to provide

reliable solutions efficiently. To overcome it, in this paper, a model based on the GCN (referred to as GCN-NPEC) proposed, and a hybrid learning strategy is proposed to train it. The trained policy can act as a heuristic that can solve problem instances rapidly.

We test this model on problem sets of different sizes with geographic information, which are generated based on the data from a real delivery system. We compare this model with some well-known existing methods. It is shown that the GCN-NPEC model improves around 5% over other existing learning models. More importantly, it outperforms a well-known heuristic solver based on OR techniques. Results also demonstrate that the GCN is effective in modeling the VRP graph, and it is important to incorporate the edge feature into the network while solving the (practical) VRP. Moreover, the joint learning strategy can make the training of the model converge fast and improve the solution quality.

Finally, we note that our GCN-NPEC model can also be applied to solve other variants of VRP, just with minor modifications. Here we discuss how to apply it to solve the pick-up and delivery (PD) problem, in which some customers require pick-up service and others require delivery service, and the vehicles can provide both services. The problem also has applications in the delivery systems of E-commence warehouses. For example, the stores have packages to be delivered by the warehouse and, in the meantime, have reusable pallets/containers or empty bottles that must be returned to the warehouse. We consider that one customer requires one or both of the services, and the vehicles still have capacities of carrying. Moreover, the vehicle starts and ends at the depot, and the picked goods are delivered to the depot. This PD problem can be solved by just using the model presented in section 3. Note that in the input features of the encoder, the demand for node i is the summation of delivery and pick-up (the value of non-required service is zero). In the decoder, the masking rules can keep the same as in §3.2.3.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[5] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*. 2702–2711.

[6] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. 2018. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 170–181.

[7] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming* 106, 3 (2006), 491–511.

[8] Bezalel Gavish and Stephen C. Graves. 1978. The travelling salesman problem and related problems. (1978).

[9] Bruce L. Golden, Subramanian Raghavan, and Edward A. Wasil. 2008. *The vehicle routing problem: latest advances and new challenges*. Vol. 43. Springer Science & Business Media.

[10] Google. 2019. OR-Tools. https://developers.google.com/optimization

[11] Gurobi Optimization, LLC. 2019. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[14] Keld Helsgaun. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University* (2017).

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] JQ James, Wen Yu, and Jiatao Gu. 2019. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems* 20, 10 (2019), 3806–3817.

[17] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227* (2019).

[18] Yoav Kaempfer and Lior Wolf. 2018. Learning the multiple traveling salesmen problem with permutation invariant pooling networks. *arXiv preprint arXiv:1803.09621* (2018).

[19] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*. 6348–6358.

[20] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*. 1008–1014.

[21] Wouter Kool, Herke van Hoof, and Max Welling. 2019. Attention, learn to solve routing problems! *In International Conference on Learning Representations* (2019).

[22] Gilbert Laporte. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 3 (1992), 345–358.

[23] Gilbert Laporte, Michel Gendreau, J-Y Potvin, and Frédéric Semet. 2000. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research* 7, 4-5 (2000), 285–300.

[24] Humberto Longo, Marcus Poggi De Aragao, and Eduardo Uchoa. 2006. Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research* 33, 6 (2006), 1823–1837.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[26] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. 2018. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*. 9839–9849.

[27] Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. 2017. A note on learning algorithms for quadratic assignment with graph neural networks. *Stat* 1050 (2017), 22.

[28] Ted K Ralphs, Leonid Kopman, William R Pulleyblank, and Leslie E Trotter. 2003. On the capacitated vehicle routing problem. *Mathematical Programming* 94, 2-3 (2003), 343–359.

[29] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[30] Paolo Toth and Daniele Vigo. 2002. *The vehicle routing problem*. SIAM.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[33] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in neural information processing systems*. 2692–2700.

[34] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[35] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[36] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* (2018).

A Experimental Results

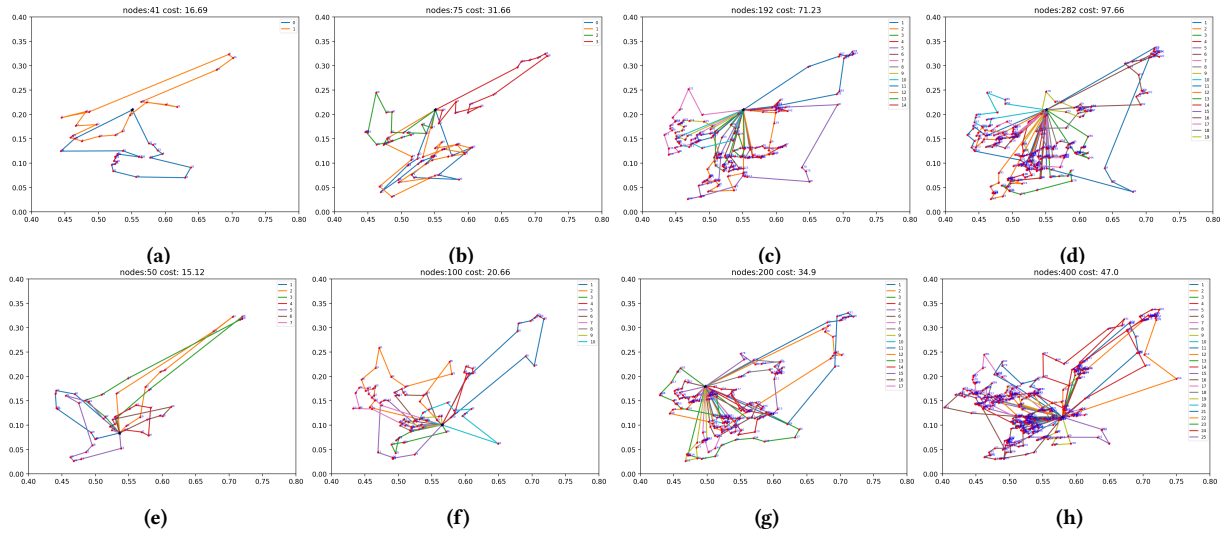


Figure 6: Example solutions by GCN-NPEC. (a), (b), (c), (d) are examples from VRP-R and (e), (f), (g), (h) are examples from VRP50, VRP100, VRP200, VRP400, respectively. The node marked with symbol '⋆' represents the depot. Legend coloring indicates different routes and the title of each graph indicates the number of nodes and the cost of the solution.