A Graph-based Model for Ride-sharing Order Dispatching via Decentralized Multi-agent Deep Reinforcement Learning

Xie Siyue
Department of Information Engineering
Chinese University of Hong Kong
Shatin, NT, Hong Kong
xiesiyue@link.cuhk.edu.hk

Li Yiming
Department of Information Engineering
Chinese University of Hong Kong
Shatin, NT, Hong Kong
liyiming@link.cuhk.edu.hk

Abstract

Ride-sharing has recently become one of the most popular ways for people to take on short-distance travelling and plays a more and more important role in modern transportation systems. By dispatching waiting orders to idle drivers, ride-sharing platforms can reallocate available transportation resource efficiently. In this paper, we build a ride-sharing simulation environment and propose a primary framework to learn the decision strategy of order dispatching for ride-sharing platforms. Different from some traditional methods, which mostly depends on a centralized decision algorithm, we propose a decentralized Multi-Agent Reinforcement Learning (MARL) strategy to build the dynamics of order dispatching for all agents. The proposed framework consists of two main modules: a Graph Convolutional Network (GCN) and the A2C framework that based on policy-gradient. The former module captures the mutual relationships among all agents in the region-of-interest, while the latter takes relationship information into account for decision making. By integrating GCN with the policy-gradient algorithm and sharing the decision strategy among all agents, the framework is able to choose a most suitable order specific for each driver in each time step. Extensive experiments are conducted on the environment that is built by ourselves, where real-world order data from Didi Haikou Order dataset is loaded in the environment. Experimental results show that the proposed GCN-based multi-agent framework outperforms several greedy methods as well as a deep-learning baseline, which demonstrate the effectiveness of our model.

1 Introduction

Peer-to-peer ride-sharing is a type of service that passengers travel in a private vehicle arranged by some qualified companies, for free or a fee. Such service is now available in many platforms, including Uber, Lyft, Grab, etc. Ride-sharing reshapes the way of people to travel and is playing a more and more important role in people's daily life. For example, by the end of 2019, the number of users of Didi Chuxing (a Chinese ride-sharing platform) has exceeded 550 million and the platform provides more than 10 billion ride-sharing service for users in a year [1].

Ride-sharing benefits our society in many aspects. One advantage is that transportation resources can be reallocated by properly matching idle drivers with waiting passengers. But in real-word scenarios, there are an enormous number of idle drivers cruising in the city, while plenty of passengers submit

[1] https://www.businesswire.com/news/home/20190710005884/en/DiDi-Introduces-New-Product-Upgrades-International-Markets

IERG 6130 Spring 2020 - Project Report.

order request to the platform concurrently. To balance the demands and supplies for both drivers and passengers, an efficient order dispatching algorithm is necessary for every ride-sharing system. However, there are many challenges to deal with ride-sharing order dispatching. On one hand, model for this task should be able to handle the problems of many-to-many matching in dynamic environment. Thousands of orders and drivers are to be matched in every minute, which can result in high computation cost if the platform conducts pair-wise matching. The information of both demands and supplies, e.g., the locations of each customer, the number of idle drivers, the price of an order, etc., changes over time, which requires the model to adaptively response to different kinds of requests in real-time. On the other hand, models are expected to find a trade-off between long-term and short-term rewards in order to maximize the profits for the platform. An order with a high price can bring a big reward to the platform immediately but may do harm to the overall profits in a long run. This is because the time cost of a specific order is usually proportional to the order price. Sometimes a driver can earn more by taking multiple short-trip orders instead of serving a long-distance request. The location related to the drivers and customers also makes a difference. For example, the driver should take a long time to pick up the customer if he/she takes an order with a remote departure location. Also, a destination with dense population can benefit the driver or the platform more than a sparsely populated area. Considering the case when there are two orders start in the same region but ends at two differen destinations: one goes to the center of the city while the other's destination is in a distant suburb. Intuitively, the driver takes the latter order may need to wait a longer time or move a longer distance to catch the next order, which prevents the driver or platform from obtaining continuous income. Such situation can get more serious in peak hours when the number of order requests overwhelms the number of available drivers in the neighborhood. As such, a proper order dispatching model should not only focus on the instant rewards given by an order but also estimate the future demand-supply gap for maximizing the overall income in a long run.

Many previous works try to solve this problem with different strategies. Traditional methods dispatch orders based on some greedy or rule-based methods, for example, assigning a nearest driver to serve a waiting passenger or choose a driver nearby that can minimize the pick-up time [1] [2] [3]. Others may resort to the queueing strategy following the principle of first-come-first-serve [4]. Although these methods are easy to implement, the order dispatching strategies they used in the model mainly focus on the satisfaction of immediate requests. Such centralized methods are unable to model the coordinations or interactions among different drivers and fail to handle long-term scheduling. In this way, some orders that are more suitable for a specific driver will be ignored, which leads to a suboptimal result when looking into a longer future. Some researchers formulate order dispatching task as a combinatorial optimization problem [5], which makes decisions in a global view. However, this kind of methods require to compute all available vehicle-order matches and rely heavily on handcrafted features. Such drawbacks result in high computation cost and prevent it from generalizing to large-scale ride-sharing platforms.

In recent years, Deep Reinforcement Learning (DRL) algorithms achieves great success in many tasks in the field of machine learning, and therefore be introduced to solve the problem of order dispatching [6] [7] [8]. By formulating the problem as a sequential decision making process, the models based on RL is able to learn a decision policy that takes the long-term rewards into account. Some researchers propose centralized controlling to make an estimation of the future rewards for each order-driver matching under specific states [6], while others build decentralized framework [7], where each driver is regarded as an agent to interact with a complex environment, to learn to make suitable decision for each driver. However, these two kind of methods are still far from perfect. For centralized methods, the estimation of pair-wise matching can lead to a high computation cost when the number of agents scales up. As for decentralized framework, interactions among agents are usually ignored [7] or only built with a subgroup, which is hard to simulate the real-world scenarios.

In order to address the aforementioned challenges, we propose a primary model that is based on A2C framework with Multi-agent Reinforcement Learning (MARL) to handle the task of ride-sharing order dispatching. A general view of the proposed method can be found in the right part of Fig 1. Similar to some previous DRL-based methods, in our framework, each vehicle/driver is regarded as an independent agent. Instead of dispatching an order to a specific driver, we learn a policy that shared among all agents to teach each driver to choose the most suitable order for themselves. Specifically, each agent is expected to take an action in each time step and trained to pick up the order that can gain the most rewards in the future. In addition, the region of interest is built as a static graph, which enables us to incorporate Graph Neural Network (GNN) into the framework to model the mutual

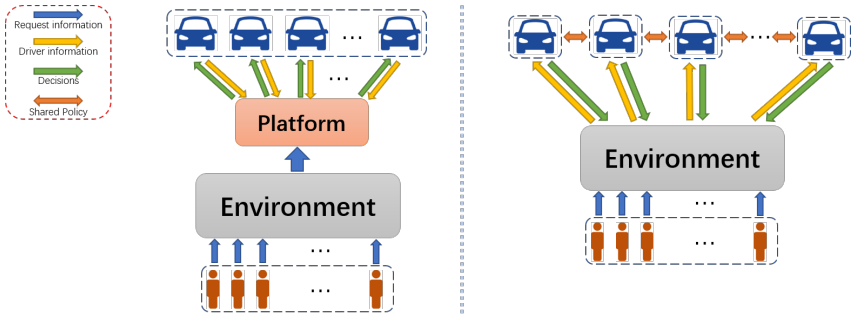

Figure 1: The general framework of traditional centralized method (left) and our proposed decentralized model (right). Centralized framework makes decisions based on a platform back-end that collects all available information, while the proposed decentralized model learns a share policy for all drivers to let them make decisions and interact with the environment by themselves.

relationships among different agents. By capturing the relationship information and distributing it to all agents, the learned policy is able to make decision for a specific agent based on the current competitive situations with others. In this way, our model can not only take long-term rewards into consideration, but also make a decision that suitable for each driver in the current time step. Different from other MARL-based method, the policy is designed with the perspective on agents instead of the platform, which can be more beneficial to drivers. As the policy works in each individual agent, the model can be implemented in the driver-end without many modification, which can also be more practical. Furthermore, in order to interact with the agents, we build a environment that simulates the real-world ride-sharing scenarios. Our model is evaluated using a real-world ride-sharing order dataset. Experimental results show the efficiency of the proposed model.

The contributions of our proposed method can be summarized as follows:

- We build a ride-sharing environment for order dispatching, which is able to simulate the real-world scenarios for the task
- We propose a decentralized MARL training framework for order dispatching. By sharing the policy among all agents, our model is able to make specific decision for each driver and reduce the computation cost.
- We incorporate the GNN into the proposed MARL framework for mutual relationship modeling among agents. By capturing the relationship information and take it into account in the decision-making process, the policy can yield a decision for each agent that balance the instant and the long-term rewards.

2 Related Works

2.1 Reinforcement Learning on Order Dispatching

Deep Reinforcement Learning (DRL) has gathered more and more attention in the past few years and it has already been successfully applied to the tasks required decision making, such as game playing [9] [10], recommendation systems [11] [12] [13], autonomous navigation [14] [15] [16], etc. Some recent works also introduce DRL to handle the task of order dispatching. Zhou et al [7] propose a Multi-Agent Reinforcement Learning (MARL) framework, which is based on the Q-learning algorithm, to model the action dynamics of all drivers. In their framework, each driver is regarded as an agent and the whole city is divided into multiple grids with equal size, where agents can move from one grid to another grid to pick up an order. In order to speed up the learning process and balance the gap between demands and supplies, they additionally introduce KL-divergence optimization into their model. However, in their model, all agents work independently, which fails to model the mutual competition in the real-word scenes. Lin et al. [8] propose a multi-agent contextual Deep-Q Network (DQN) to coordinate the actions among different drivers. Meanwhile, a contextual actor-critic framework is applied to the learning process, which adapts the model to the dynamically changing action space and therefore keeps a more stable performance. Xu et al. [6] model the problem

as a Markov Decision Process (MDP). Different from other MARL-based methods, the model is implemented with a centralized controlling strategy. They first conduct a offline learning step to learn a value function that quantized the demand and supply patterns. An online planning step is then conducted to estimate a matching value for each driver-order pair. The order dispatching problem is finally solved using a combinatorial optimization algorithm. Li et al. [17] propose a MARL model based on Actor-Critic framework. The model they proposed goes with a centralized training but with a decentralized execution. Also, they incorporate the Mean Field Approximation (MFA) into their framework, which allows an agent to interact with others within a fixed-size neighborhood. Following the work of the aforementioned methods, we build a MARL model with the A2C framework. Instead of training in a supervised manner, we learn a policy and share it among all agents, which enables a driver to choose a more suitable order in each single time step.

2.2 Graph Neural Network

Graph Neural Network (GNN) is a family of models that are used to process unstructured data. Various works have explored the effectiveness of GNN, which greatly improve the research on relationships inference. For example, Kipf et al. [18] simplify the spectral graph convolution using a first-order approximation, which significantly improves the performance on the task of semi-supervised node classification. Hamilton et al. [19] propose a inductive model named GraphSAGE to generate node embeddings through spatial aggregation and successfully implemented in some large-scale graphs. Some methods also joint RL with GNN. In the work of [20], the traffic dynamic of an Optical Transport Network (OTN) is modeled using a GNN, while DQN is introduced into the system for routing policy decision. Hamrick et al. [21] introduce GNN to solve the gluing task. In their model, the inter-connection among different blocks are modeled as graph, while representations of each node and edge will be taken as input into a DQN for gluing decision. Chen et al. [22] integrate GNN with Q-network to handle the task of structured dialogue policy optimization. Instead of using a Multi-Layer Perceptron, the authors embed GNN into the Q-network, which enables the model to make relationship inference when yielding dialogue actions. Inspired by the previous works, we introduced GNN into our proposed MARL framework for relationships modeling. By capturing the relation information and share it among all agents, our method is able to estimate the competition level of each order, which leads the policy to make a more reasonable decision for each agent in each time step.

3 Environment Setup

In this paper, we simplify the problem of order dispatching and formulate it as a sequential decision-making process and our goal is to maximize the cumulative rewards of the process over a certain time period for each driver. To learn a policy that can adapt with such a process, we build a specific environment to interact and evaluate our model. The environment simulates the real-world order dispatching scenarios, where some core information of the environment used used to optimize our model can be summarized into a tuple as follows:

$$\Gamma =< \mathcal{G}, \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma > \tag{1}$$

Here, $\mathcal{G}$ indicates a graph structure that represents the region we are interested in, $\mathcal{D}$ is the set of agents, $\mathcal{S}$ is the set of states of the environment, $\mathcal{A}$ is the set of actions that taken by agents, $\mathcal{R}$ is a set that records the reward of each action and γ is a discount factor for cumulative reward computation. In the following, we give the detailed description of our environment setup and the related components.

Graph Construction Many previous works [6] [7] [8] separate the whole city into multiple grids with same shape in order to model the regions that drivers and customers will occur. Staying different from these approaches, in this work, the whole city in the environment is represented as a undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$ is the set of nodes and edges. An example of the graph is shown in Fig 3. In this work, each node of the graph covers a region and the edge between two nodes assumes that there is a road link between the two corresponding regions. To match the graph with the geographical area we are interested in, we use K-means clustering to generate the nodes we need. Specifically, we collect a set of historical ride-sharing orders over a month, which record the departure and destination locations. K-means is then utilized to cluster all the collected orders based on the departure and destination locations. The outputs of K-means, i.e., a set of clustering centers,

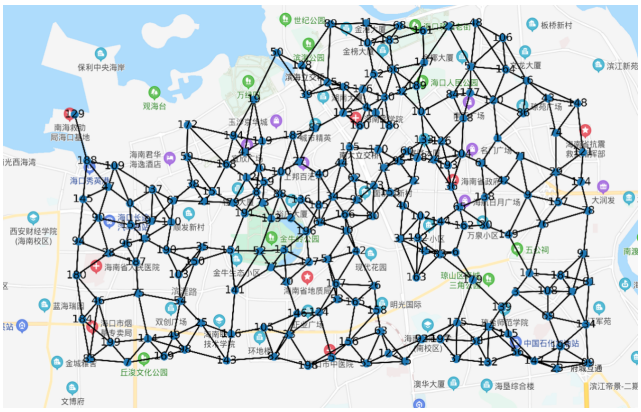

Figure 2: The geographical area we are interested in. The whole city is represented as an undirected graph, where each node covers a region with different shapes. An edge is built between nodes, which assumes that there is a road link between these two regions.

are treated as the nodes of the graph. For a specific order, we map the departure/destination location to a node that nearest in distance. For a node in the graph, we connect it with k (we set $k = 5$ in our implementation) nearest nodes but constrained with a distance threshold, which forms the edge set. As such, we can build a graph that matches the real-world regions. In addition, we calculate some statistics of each node based on historical order records, e.g., the averaged order requests occur in a node in an hour, and load it to the graph as node attributes. The location information of each agent is also loaded in the graph. Compared with typical grid-setting, advantages of representing the city as a graph are obvious. On the one hand, each node covers a region with different areas and shapes, which results in a more detailed description of order distributions. For example, there will be more nodes in request-frequent areas like city center than that of the suburban where request can be sparse. Such setting segments the city in a more fine-grained manner based on the order density, which makes it more easier for our model to learn the order distribution and predict the trend of order request in the near future. On the other hand, each node covers a region with different shape, which can be more flexible than that of grid-region. One can even evolve the graph topology by continuously update the order records that used in clustering. Also, graph structure setting enables us to incorporate GNN method into our model, which make it possible to model the mutual relationships for different drivers/agents.

Agent In this work, each driver is regarded as a agent. Each agent can either be idle or busy, while only idle agents are able to receive order requests and take actions. In our environment settings, the number of agents is fixed, i.e., $|\mathcal{D}| = N_{\mathcal{D}}$, where $N_{\mathcal{D}}$ is a constant. Since the goal of each driver is the same, i.e., maximize the overall rewards after a certain period of time, we treat all drivers as homogenous agents. Thus, we only need to learn a policy that share among all agents, which avoid heavy computation cost.

State As shown in Fig. 1, agents in our proposed model can directly interact with the environment. For the d-th agent, the state observed from the environment in the t-th time step can be formulated as $\boldsymbol{s}_{d,t} \sim< \mathcal{O}_{d,t}, \mathcal{G} >\in \mathcal{S}$, where $\mathcal{O}_{d,t}$ represents a list of candidate orders. Each order in the list contains the information of departure and destination locations (represented by the node index), order price and the estimated time cost to finish the order (with regard to a specific agent). In order to be compatible with the proposed policy that with fixed output space, the environment samples a fixed number of orders for each agent and load it to $\mathcal{O}_{d,t}$. To be specific, order requests are submitted in each time step in different locations and collected by the environment. Intuitively, it will be more efficient for a driver to take an order occurs nearby. Therefore, for a specific agent, the environment samples N_o orders from all order requests, where each order follows a sampling probability that inversely proportional to the distance between the location of the agent and the order departure. In this way, the orders nearer to an agent are more likely to be sampled to the order list.

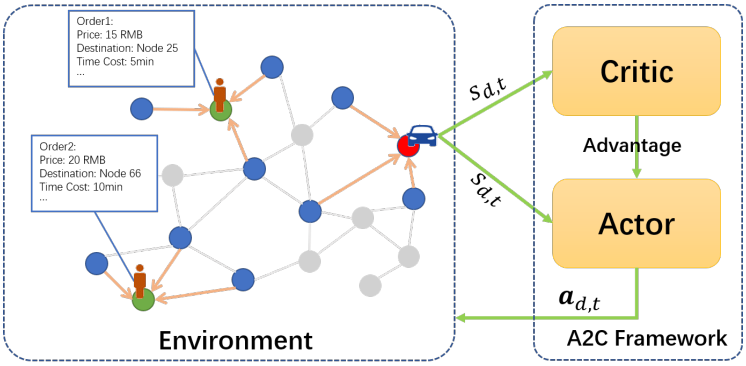

Figure 3: The architecture of the proposed model, which consists of a GCN module and an A2C framework. The GCN module aggregates neighborhood information for relationships modeling, while A2C is trained to make decisions for each agent.

Action The agent under our environment settings can choose to take a order or not. The action for the d-th agent in the t-th step can be represented as a one-hot vector $\boldsymbol{a}_{d,t} \in \mathcal{A}$ with $(N_o + 1)$ dimension, where the first N_o dimension correspond to the N_o orders in the order list, while the additional dimension indicates the agent will keep idle. Note that only idle agents can receive meaningful state from the environment and take an action. For agent who chooses to keep idle, it will cruise to one of its neighboring nodes in the next time step. For agent who chooses to take an order, the status of it will turn to 'busy' immediately and become 'idle' again after it arrives at the destination of the order. The required time it travels to the destination is given by the time cost in the corresponding order state.

Reward Following the goal of our model, we directly use the order price as the reward in each time step. For agents who choose to keep idle, zero reward will be returned by the environment.

Competition Rule It is intuitive that an order occurs nearby with a high price can be attractive to a driver. However, when there are multiple drivers in the region, only one of them can win the order. In order to simulate such situation in the real-world scenarios, we introduce a competition rule into the environment. For agents who choose the same order, rewards can only be assigned to one of the agents. Specifically, the probability of winning the order is inversely proportional to the distance between the location of the agent and the departure. By introducing such a competition rule, interactions among agents can be built. Also, the proposed model will be forced to learn the competition status in the environment and estimate the risk of taking an order, which makes it more reasonable to real-world practice.

4 Proposed Method

In this section, we describe the technical details of our proposed method. The architecture of the model is shown in Fig. 3. In general, the model consist of two main modules, i.e., a GCN module and an policy network based on A2C framework. The former module is responsible for relationships modeling based on the graph structure, while the latter one takes node representations (generated from GCN) and order information as input for decision making. By combining GCN with the A2C framework, our model is able to generate a proper action specific for each agent, which makes it more effective to solve the order dispatching problem.

4.1 Relationships Modeling Using GCN

As we mentioned in Section 3, the whole city is modeled as a undirected graph in our environment. To deal with such a non-Euclidian data structure, we incorporate the GCN framework into our model.

The GCN used in our model follows the work of Hamilton et al. [19]. In this task, the GCN module is used to generate a vectorized representation of each node, which quantizes the comprehensive status

of a specific region. Specifically, given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, the operation in a GCN layer can be formulated as follows:

$$h_v^l = \sigma(W_1^l h_v^{l-1} + W_2^l \cdot \tilde{h}_v^l) \tag{2}$$

where W_1^l and W_2^l are two learnable weight matrix for linear transformation in the l-th layer, h_v^l is the embedding of node $v \in \mathcal{V}$ in the l-th layer and $\sigma(x)$ is an activation function. $h_{\tilde{v}}^l$ stands for the aggregated information from the neighborhood of node v, which can be computed by:

$$\tilde{h}_v^l = \frac{1}{|N(v)|} \sum_{u \in N(v)} h_u^{l-1} \tag{3}$$

where $N(v)$ is the neighborhood of node v and $|N(v)|$ is the number of nodes in the neighbor of v. As for the first layer, the embedding of the v-th node (i.e., h_v^{l-1}) is initialized as a vector of raw node attributes, which are some statistics we mentioned in Section 3 and some features that reflect the dynamic of the current environment (e.g., the number of idle agent in this node, the number of order request in this node, etc.). By recursively stacking L GCN layers, we can build a deep network that is powerful to generate high-level representation for a node (region).

It can be observed that the nature of a GCN layer is to collect the information from the one-hop neighbor of a specific node. However, by recursively stacking L GCN layers, we can receive the information from the L-hop neighbors, which makes it possible for our model to build up the relationships between two regions that are far from each other. In our model, the raw node attributes are order statistics that calculated from history records. Thus, the GCN model not only able to capture the local features of a specific node, but also the trend of order requests in its nearby. Such information can be helpful for the decision policy to look further. For example, the long-term rewards of an order that ends at the center of the city is more likely to be higher than that of an order that ends at suburban. This is because the order requests in center of the city could be much more than that of suburban, which results in a much smaller waiting time to pick up the next order. Therefore, by incorporating the node representation into decision making, the policy is able to predict the near future and yield a more reasonable action that can maximize the overall rewards of an agent.

Another advantage of using GCN is that we can quantize the interactions among agents. Since the location of each agent is bound with a node and the graph is shared among all agents, GCN is able to capture the mutual relationships of different agents by generating node representations. Compared with some MARL works [23] that need to build a communication mechanism among agents, using a GCN model to model the interactions among agents is more efficient and with lower computation cost, which makes the learned policy perform better in order dispatching task.

4.2 Observation Preprocessing

After obtaining node embeddings by the GCN model, we need to learn a policy that can make decisions for each agent. To make it easier for the model to make a proper decision, we additionally collect some information about each order and each agent, which will be packed together with node embeddings to train the policy network. Specifically, for a particular agent, the node embedding of the region that the agent located will be collected. Geographical information that related to a specific order, i.e., the node embedding of the departure and the destination places, will also be collected. In order to distinguish different orders, the estimated time cost and price for each order will be extracted from the order list $\mathcal{O}_{d,t}$ at the t-th time step. All these data will be concatenated as a single vector to represent the state ($\boldsymbol{s}_{d,t}$) and taken as input to the policy network for decision making.

4.3 Policy Learning with A2C Framework

The goal of our policy network is to maximize the discounted overall rewards in an episode for each agent, where the objective function can be formulated as follows:

$$\mathbb{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, d \sim \mathcal{D}}[\sum_t \gamma^t r(\boldsymbol{s}_{d,t}, \boldsymbol{a}_{d,t})] \tag{4}$$

where τ indicates an episode, π_θ is the policy parameterized by θ, γ is the discounted factor and $\mathcal{D}$ is the set of agents we mentioned above.

To this end, we utilize the framework of A2C [24] to optimize our models. In our model, the actor and critic network are both constructed by a two-layer MLP model with similar settings.. The difference

Algorithm 1 Training steps of our model

Initialization of actor parameter θ and critic parameter ϕ, learning rate α
for $iteration = 1, 2, \ldots$ **do**
  for each step t in an episode **do**
    generate $\boldsymbol{s}_{d,t}$ with GCN module
    generate $\boldsymbol{a}_{d,t}, r_{d,t}$ following policy π_θ
    generate $\boldsymbol{s}_{d,t+1}$ with GCN module
    $\delta(\boldsymbol{s}_{d,t}) = r_{d,t} + \gamma V_\phi(\boldsymbol{s}_{d,t+1}) - V_\phi(\boldsymbol{s}_{d,t})$
    $\phi \leftarrow \phi + \alpha \frac{1}{|N_\mathcal{D}|} \sum_{d \in \mathcal{D}} \delta(\boldsymbol{s}_{d,t}) \nabla V(\boldsymbol{s}_{d,t+1})$
    $\theta \leftarrow \theta + \alpha \frac{1}{|N_\mathcal{D}|} \sum_{d \in \mathcal{D}} \delta(\boldsymbol{s}_{d,t}) \nabla \log(\pi_\theta(\boldsymbol{a}_{d,t}|\boldsymbol{s}_{d,t+1}))$
  end for
end for

of this two network locates in the output space. The output of the actor network is a vector $\boldsymbol{a}_{d,t}$ that with $|\mathcal{O}_{d,t} + 1|$ dimension, where the first $|\mathcal{O}_{d,t}|$ values in $\boldsymbol{a}_{d,t}$ are the probabilities of the agent to choose the corresponding candidate orders and the last dimension is the probability of keep idle in the t-th step. As for the critic network, it only output a scalar value. Compared with combinatorial optimization method, A2C can further consider long term rewards. Therefore, the model can make a balance between short-term and long-term rewards when making decision.

In our task, the rewards are the actual price of the order provided in the data, which are always positive. It may cause large variance when the model tries to estimate how good itself to satisfy the current 'positive rewards'. To reduce the variance of our policy network, the critic network will estimate the expected rewards of taking a specific order. Then we subtract the estimated rewards value from the current rewards, which is similar to A2C. The advantage of introducing such a critic network is obvious. For example, consider an order with the destination far away from city center. It is intuitive that order requests could be sparse in such a region. In this case, taking this order contributes little to the future rewards or even do harms to overall rewards (as the driver may take a long time to pick up the next order). By using critic network to estimate the expected rewards of current observation, the actor network can make a decision that more suitable for the agent. Formally, the critic network computes TD error by the following equation, which is an unbiased estimate of the advantage function.

$$\delta(\boldsymbol{s}_{d,t}) = r_{d,t} + \gamma V_\phi(\boldsymbol{s}_{d,t+1}) - V_\phi(\boldsymbol{s}_{d,t}) \tag{5}$$

where V_ϕ is the critic network parameterized by ϕ. $\delta(\boldsymbol{s}_{d,t})$ represents TD error, which is the gap between the immediate reward and the estimated reward. We then replace the returns in the original objective function of policy-gradient by the TD error, which acts as the advantages in A2C. Therefore, the objective function in Formula 4 can be modified as follows:

$$\mathbb{J}(\theta, \phi) = \mathbb{E}[\log \pi_\theta(\boldsymbol{s}_{d,t}, \boldsymbol{a}_{d,t}) \delta(\boldsymbol{s}_{d,t})] \tag{6}$$

In our training stage, we optimize the parameters of our model by maximizing the objective function in Formula 6. We summarize the steps to train our policy network, which is shown in Algorithm 1.

5 Experiments

Our experiments are conducted using a real-world ride-sharing order dataset. We also compare our model with some greedy methods as well as a baseline method that without graph information. In the following of this section, we will first introduce the dataset and experimental setups. Then, experimental results and analysis will be presented.

5.1 Dataset

Didi Haikou Order The Didi Haikou Order dataset [2] is a public ride-sharing order dataset released by Didi Chuxing. There are more than 10 million order record in this dataset, which are collected in the city of Haikou from May 1st to Oct 31st in 2017. We select the orders that start and end in

[2] `https://outreach.didichuxing.com/app-vue/dataList`

Table 1: Experimental Results on Didi Haikou Dataset (Averaged Cumulative Rewards per agent)

Algorithm	Single Agent Setting	Multi-agent Setting
RandomPolicy	226.1	234.0
PricePolicy	266.3	**280.3**
TimePolicy	239.8	226.1
UnitPolicy	278.1	269.4
DeepPolicy	311.6	254.3
GCNPolicy	**337.3**	243.7
DeepPolicy (Fine-tuned)	-	279.2
GCNPolicy (Fine-tuned)	-	**305.5**

the four main districts of Haikou (i.e., Xiuying district, Longhua district, Qiongshan district and Meilan district). Therefore, 4,988,540 orders in total are involved in our experiments. We clean up the raw data and the remaining attributes for each record entry include time stamp, departure location, destination location, estimated travel time and distance from departure to destination and the order price.

5.2 Experimental Setups

We split the time interval (i.e., from May 1st to Oct 31st in 2017) into multiple windows, each of which last for 2 minutes. Each time window is corresponding to a 'time step' in our method. «««< HEAD Orders that are submitted within the same time window will be gathered together and then loaded to the environment for dispatching. In our experiment, each episode is made up of 240 time steps, which corresponds to an 8-hour time interval. Each episode has a two-hour overlap with its previous and subsequent episode. Therefore, there are totally 3111 episodes after preprocessing, 80% of which are randomly selected to form the training set and the remaining are treated as testing set.

In our experiments, we build a graph with 400 nodes for the environment (i.e., the number of clusters in k-means clustering is 400) and the length of candidate order list is 5 ($|\mathcal{O}_{d,t}| = 5$). Each agent is randomly assigned to a node in the first time step in the training and testing stage. For the network architecture of the proposed model, we build a two-layer GCN, which is followed by a one-layer fully-connected neural network. The hidden dimension of each layer are all set to be 128. The discount factor $\gamma = 0.99$.

We compare our model with four heuristic/greedy policies and one deep-learning baseline. The detailed information of each compared model is described as follows:

- **RandomPolicy** This method randomly selects an order at each time step.
- **PricePolicy** This baseline method select the order with highest price value at each time.
- **TimePolicy** This model selects the order that with minimal time cost.
- **UnitPolicy** This method makes a trade-off between the PricePolicy and the TimePolicy. Specifically, agents choose an order that with highest price reward per time cost unit.
- **DeepPolicy** Different from the policies aforementioned, this method make decisions based on a deep neural network. The framework of this policy is similar to our proposed model. However, the GCN model is replaced by a two-layer fully-connected network, with the raw node attributes as input.

For illustration convenience, the proposed method is denoted as GCNPolicy in our following discussions.

All our models are implemented on PyTorch [25] and DGL [26]. All the experiments are conducted on a single Linux machine with CPU AMD Ryzen Threadripper 3970X and 2 NVIDIA RTX TITAN GPUs. We report the Averaged Cumulative Rewards (ACW) in an episode of each agent in the following.

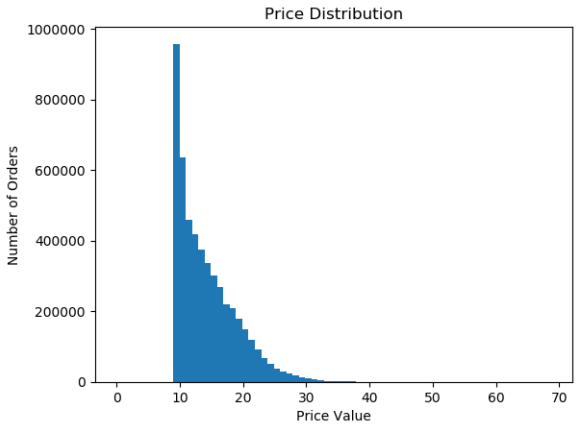

Figure 4: The distribution of price values of orders that are used in our experiments. It can be clearly observed that orders with low price value (value smaller than 20) dominate the whole dataset.

5.3 Experimental Results

We conduct three experiments to evaluate our model. To be specific, we first train and test our model with single agent. Then, experiments with multi-agent settings are conducted. Since the policy is shared among all agents, we conduct another trial that we fine-tune our multi-agent model by using the policy trained under single agent settings.

5.3.1 Order Dispatching with Single-agent

The experimental results under single agent setting is shown in the second column of Table 1. We can clearly observe that the proposed method (GCNPolicy) outperforms all other algorithms by a large margin. It can be seen that the performance of TimePolicy is similar to the RandomPolicy, which means that the factor of time cost may not influence much under such environment setting. In contrast, the greedy policies that take price value into account (i.e., PricePolicy and UnitPolicy) perform much better, which means that the price of each order can be crucial to the overall outcome. We can get a more clear view on this fact by the distribution of order price, which is shown in Fig. 4. It can be observed that the orders with low price (i.e., price value smaller than 20 RMB) dominate the dataset. Although the price value of each order is proportional to the actual time cost, the difference of time cost among low price orders can be very marginal. Since agents can only make decision for every 2 minutes (each time step covers a 2-minute time window), the marginal difference in time cost may be covered by the time window. In such a situation, choosing an order with higher price can benefit the agent more in the current step.

Compared with the above greedy methods that focus more on short-term rewards, learning-based methods (i.e., DeepPolicy and GCNPolicy) perform much better. This fact reflects that DeepPolicy and GCNPolicy can distinguish different orders and estimate the future, which help them make proper decisions adaptively in each time step. Such characteristics benefit them a lot on long-term rewards. In addition, it is worth mentioning that the proposed GCNPolicy is able to get a higher reward than that of DeepPolicy. This owes to the introduction of the GCN framework. By incorporating GCN into the model, an agent is able to collect more information from its neighbors instead of the information of the region it locates in. In this way, agent with GCNPolicy is able to make a more comprehensive analysis at each step, which results in a more superior performance.

5.3.2 Order Dispatching with Multi-agent

Since the leaned policy can be shared among different agents, we conduct an experiment that under multi-agent setting. As time and computation resource limited, we only evaluate our model using three agents. The quantitative results are shown in the third column in Table 1. Similar to the results on single agent setting, greedy policies that based on order price values perform better than other greedy methods. However, it is surprise that there is a huge performance drop appears on DeepPolicy and GCNPolicy. The averaged cumulative rewards of these two learning-based methods are still

higher than RandomPolicy and TimePolicy, but much lower than that of PricePolicy and UnitPolicy. One possible reason is that the introduction of competition rule in the environment makes it hard to train the model from scratch when under multi-agent setting. As we mentioned in Section.3, only one agent can finally win an order if there are multiple agents choose to take the order. For an agent who loses in the competition, it receive zero reward at that time step even though the order fully matches the requirement of the agent. However, by the nature of policy-gradient, the probability of an action with low reward will be reduced in the training stage. As the model is trained from scratch, the model may get confused before learning enough knowledge about the competition rule and how to make a proper decision, which results in the degrade of the performance. Another explanation is that the model is prone to act conservatively under the competition rule. It is intuitive that an profitable order is always with high risk, which may result in getting zero rewards if the agent chooses to join the competition. To avoid getting nothing in the current step, the policy may always choose a suboptimal order instead of the optimal one but with high risk. In this way, the performance of these two learning-based method can drop dramatically.

5.3.3 Fine-tuning Multi-agent Model with Pretraining

To speed up the training process and avoid of being conservative, we transfer the knowledge learned under single agent setting to the multi-agent model. Specifically, we fine-tune the single-agent model under the multi-agent setting. The results are shown in the bottom of the third column in Table 1. Compared with the model without pretraining, the performance of DeepPolicy and GCNPolicy are improved by a large margin. Also, we can find that the fine-tuned GCNPolicy outperforms all other methods. This means that the proposed model still makes sense for the order-dispatching task. As the single-agent model is trained without competition, it can fully learn the knowledge on how to properly choose an order (to maximize its cumulative rewards). By transferring the learned knowledge into the multi-agent model, in the fine-tuning stage, the GCNPolicy can focus more on learning how to balance between high rewards and the high risk instead of learning how to choose an order, which reduces the confusion and makes the training easier. Although the averaged cumulative rewards of the multi-agent model for each agent is lower than that of the single-agent model, the Gross Merchant Volume (GMV, $GMV = ACW \times N_D$) of the multi-agent model is much higher. Therefore, it will be more effective for the ride-sharing platform to implement the multi-agent model. In our expectation, GMV will increase but ACW for each agent will decrease if we continuously increase the number of agents. However, there should be a equilibrium point when GMV will increase no more. As time limited, we do not conduct further experiments to explore the equilibrium point, which can be our future work.

6 Conclusion

In this paper, we propose a MARL model for ride-sharing order dispatching based on A2C framework. In our model, each driver is regarded as an independent agent but the learned policy is shared among all agents. Thus, the model is able to learn different knowledge from different drivers, which makes it more robust and effective in making decisions, In addition, we represent the whole city as a graph, where each node covers a specific region. Such setting enables us to incorporate GCN into our model. By recursively aggregating the topological and statistical information from the neighborhood of a node, GCN can extract high-level semantic features for each region and capture the relationships among all agents. This makes it easier for our method to model the interaction among different agents. By integrating the GCN module with the A2C framework, each agent is induced to take a proper action that is beneficial to its long-term rewards, which makes it more effective for the order dispatching task. Our model is evaluated in the Didi Haikou Order Dataset and compared with some heuristical/greedy method as well as a deep-learning based baseline. Experimental results demonstrate the superiority of our model.

References

[1] Ziqi Liao. Real-time taxi dispatching using global positioning systems. *Communications of the ACM*, 46(5):81–83, 2003.

[2] Der-Horng Lee, Hao Wang, Ruey Long Cheu, and Siew Hoon Teo. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record*, 1882(1):193–200, 2004.

[3] Bin Li, Daqing Zhang, Lin Sun, Chao Chen, Shijian Li, Guande Qi, and Qiang Yang. Hunting or waiting? discovering passenger-finding strategies from a large-scale real-world taxi dataset. In *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 63–68. IEEE, 2011.

[4] Rick Zhang and Marco Pavone. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. *The International Journal of Robotics Research*, 35(1-3):186–203, 2016.

[5] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2151–2159, 2017.

[6] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913, 2018.

[7] Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2645–2653, 2019.

[8] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783, 2018.

[9] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[10] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, et al. Mastering complex control in moba games with deep reinforcement learning. *arXiv preprint arXiv:1912.09729*, 2019.

[11] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1187–1196, 2018.

[12] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. *arXiv preprint arXiv:1812.10613*, 2018.

[13] Tiago Wiedmann, Jorge Luis Victória Barbosa, Sandro José Rigo, and Débora Nice Ferrari Barbosa. Recsim: A model for learning objects recommendation using similarity of sessions. *J. UCS*, 22(8):1175–1200, 2016.

[14] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[15] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[16] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6629–6638, 2019.

[17] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference*, pages 983–994, 2019.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[20] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: An optical network routing use case. *arXiv preprint arXiv:1910.07421*, 2019.

[21] Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

[22] Lu Chen, Bowen Tan, Sishan Long, and Kai Yu. Structured dialogue policy with graph neural networks. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1257–1268, 2018.

[23] Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation science and engineering*, 7(3):607–616, 2009.

[24] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[26] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*, 2019.