Optimizing Large-Scale Fleet Management on a Road Network using Multi-Agent Deep Reinforcement Learning with Graph Neural Network

Juhyeon Kim
Department of Electrical and Computer Engineering
Seoul National University
1 Gwanak-ro, Gwanak-gu, Seoul 08826
cjdeka3123@snu.ac.kr

November 12, 2020

Abstract

Optimizing fleet management is an important issue in ride-hailing service in terms of customer's satisfaction and increased revenue for drivers. However finding an optimal strategy in a real environment that demand and supply are changing in real time, is a challenging problem. In this paper, we try to solve this problem by using multi-agent reinforcement learning. Instead of grid which was used in existing works [11, 9, 26, 5], we use a graph to represent a road network more realistically. We model the problem using Markov game and adopt stochastic policy update which can resolve the problem of greedy policy update in multi-agent scheme. We use modified DQN [13] to fit our problem. To approximate Q values on the graph, we use two basic graph neural networks, GCN [7] and GAT [19]. We design a simulator using real taxi call data and evaluate the algorithms under various conditions. The result demonstrates effectiveness of the proposed stochastic policy update model with graph neural network.

1 Introduction

Optimizing vehicle dispatching in ride-hailing service is important. Without an efficient allocation, customers will not be able to use the service due to the lack of drivers in some areas, while many drivers will remain idle in other areas. Optimized fleet management can match more customers and drivers, and it would be desirable for the both. Many online ride-hailing platforms have large amounts of cumulative demand/supply data, but using these data to design an optimal vehicle management system is a difficult task. The difficulty comes mainly from the dynamic nature of the problem that current relocation policy also affects future demand/supply. Therefore, instead of traditional supervised learning, reinforcement learning(RL) that agents interact with then environment and learn an optimal policy, has been widely adopted to solve the fleet management problem [4, 22]. However, due to the high dimensionality and complexity of the problem, there are limitations in applying traditional RL methodologies. Recently, deep reinforcement learning(DRL) has been introduced and shown brilliant performance in many decision making problems [12, 17]. However still we cannot directly apply DRL to the fleet management problem because of the scale of the problem. We have to control a large number of drivers approaching tens of thousands, which has been rarely studied. There are many works [15, 14] that had dealt with multi-agent RL(MARL), but most of them only considered at most few hundred of agents. Several recent studies [11, 9, 26, 5] have suggested a method to apply multi-agent DRL(MADRL) framework for the fleet management problem and have successfully tackled real-world scenario. However, these studies assumed grid-shaped spatial environment, which is far from the real road network. It can be more precisely modeled using a graph.

In this paper, we propose an efficient fleet management system on a road network using MADRL. Our study is basically based on aforementioned papers [11, 9, 26, 5], while have extended the problem to the graph setting. Our major contributions are listed as follows:

- We assume more realistic situation; we use a road network graph instead of a grid which has been used in previous works [11, 9, 26, 5].
- We propose large scale multi-agent RL settings on a graph. We design proper Markov Game model and propose efficient near-optimal policy finding strategy using stochastic policy update. We also show that graph neural networks can be successfully used to approximate Q values on the graph.
- We design a realistic simulator using real data and compared several algorithms. We find that our proposed method shows better performance than the others.

2 Related works

Large scale fleet management problem has been studied in many previous works. For rule-based approaches, there are greedy matching [8], centralized combinatorial optimization [25], collaborative dispatch with decentralized setting [16], or adaptive multi-agent scheduling system [1]. [23] and [21] adopted RL-based approach, but they assumed single-agent setting which cannot model complex interactions of drivers and orders.

Many recent studies are using MARL framework. [11] proposed a contextual MARL that considers geographic and collaborative contexts to prevent invalid actions. [9] suggested cooperative MARL algorithm which adopted mean field approximation to model agents' interaction. [5] introduced delayed order matching and established two-stage framework that utilizes both MARL and combinatorial method. [26] proposed decentralized system based on MARL with KL-divergence optimization. However these studies assumed a grid world system, which is far from the real road. Meanwhile, there are several studies that dealt with graph combinatorial optimization with [6, 2] or without RL [3], but large-scaled problem such as fleet management on a graph structure, has been rarely studied.

3 Problem statement

In this paper, we focus on solving a fleet management problem on a road network. Instead of gross merchandise volume(GMV, the price sum of all served orders) which has been used in previous works [23, 11, 9], we use total order response rate as an objective. We think this is more reasonable because GMV does not take account of customer's satisfaction. We use similar problem setting and notation to previous works [5, 20, 11, 23, 9]. The most significant difference is a spatial setting: we use a graph which is much more realistic than the grid in previous works. For temporal setting we use similar setting, dividing a day to T time steps. First, let's see how the problem can be formulated on a graph setting.

3.1 Problem setting on a road network

The road network can be represented as a directed graph $G_R = (V, E)$ where V is the set of intersections n and E is the set of roads l. Roads are directional, so even adjacent lanes are considered different if they have different directions. Drivers are distributed throughout the road network (Fig. 1). Note that they are not on the nodes, but on the edges(roads). Their position can be represented by road index and relative position at that road. At each time step, drivers move forward at their current road by the speed of the road. Staying at current position is not allowed. If the driver cannot go over the end of current road in this time step, it is considered as a *non-controllable* driver. If the driver can reach the end point of the current road, it is regarded as a *controllable* driver who can move to following roads. At each time step, orders are generated at each road and are assigned to idle drivers at the same road. For simplicity, we do not consider order's relative position on the road or assigning order to a driver at opposite side of the road. Our goal is to find an optimal relocation strategy for controllable drivers that can maximize order response rate. To resolve this problem, we use multi-agent reinforcement learning as previous works [11, 9, 26, 5]. Now let's design a Markov Game model for this graph setting.

3.2 Markov game modeling

We define a Markov game $G = (N, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ as following. $(N, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ refers to the total agent number, set of the state, joint action space, transition probability, reward function and discount factor.

- Agent : We define an agent as a driver who is at idle state(not serving an order). An agent can be either controllable or non-controllable. To make problem simpler, we assume that agents at the same road at the same time, are homogeneous regardless of their relative position on the road. Let's say N_t^c, N_t^{nc}, N_t as a number of controllable, non-controllable, total agents at time t. If the subscript

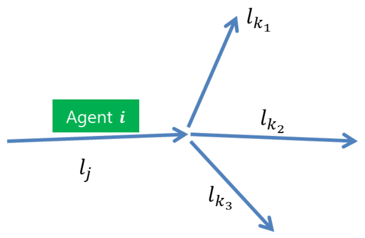


Figure 1: Example of road network. Agent i is on road l_j. At next time step, it moves forward on l_j. If it can go over the end of l_j, it can go to its neighboring roads l_ks.

j is added, think of it as the value for road l_j. Also from now, things about the agents will be represented by superscript i.

- State $s_t \in \mathcal{S}$: We use global state s_t at each time t. We observe three values at each road l_j at time t: number of idle drivers or agents $N_{j,t}$, number of calls $N^{call}_{j,t}$ and speed $speed_{j,t}$. Then we concatenate the information to get global state $s_t \in \mathbb{R}^{N_{road} \times 3}$ where N_{road} is a total number of roads. This is used as an input for a graph neural network.

- Action $a_t \in \mathcal{A}$: As mentioned before, controllable agents can be moved to the road that follows. Let's define which road to move as an action. Then, each road has a different set of possible actions. For non-controllable agents, there is no freedom to select an action. We aggregate every drivers' action to define joint action $a_t = [a^i_t]^{N_t}_1$. For simplicity, we represent action moving from l_j to l_k as $l_j \to l_k$.

- Reward $\mathcal{R}_t$: ith agent's reward at time t, R^i_t is simply set to 1 if order is assigned to agent unless 0, after the action a^i_t. Non-controllable drivers can also serve calls and get reward.

- State transition probability $\mathcal{P}$: We assume that action is deterministic. However driver number increases or decreases by random process, so driver number at each road does not change in deterministic way.

We can write ith agent's discounted return at time t as $G^i_t = \sum_{k=0}^{\infty} \gamma^k R^i_{t+k}$. The main purpose of each agent is to maximize its own expected discounted return $\mathbb{E}[G^i_t]$. From homogeneous assumption, $\mathbb{E}[G^i_t]$ only depends on agent's current road l^i_t, time step t and its controllability. Let's marginalize $\mathbb{E}[G^i_t]$ over controllability and get state-value function $V(l^i_t, t)$. Let's define a policy $\pi_{j,t}$ at road l_j as a probability distribution of moving to the following road and represent it as $\pi(l_j \to l_k, t)$. Note that policy is valid for only controllable agents and can have different dimension for each road. Define joint policy π_t at time t as $[\pi_{j,t}]^{N_{road}}_1$. Here, π_t is computed from applying a function π_ϕ to global state s_t. Assuming every controllable agent moves based on $\pi_t = \pi_\phi(s_t)$ at each time step t, we can represent state-value function and action-value function as following.

$$V_{\pi_\phi}(l_j, t) = \mathbb{E}_{\pi_\phi}[G_t^i | \text{agent } i \text{ at } l_j \text{ at time } t \text{ and follows policy from } \pi_\phi] \tag{1}$$

$$Q_{\pi_\phi}(l_j \to l_k, t) = \mathbb{E}_{\pi_\phi}[G_t^i | \text{agent } i \text{ moves } l_j \to l_k \text{ at time } t \text{ and follows policy from } \pi_\phi] \tag{2}$$

Here, as [11] did, we also assume that Q value only depends on the destination road. To distinguish this value from Q, let's represent it with Q_V: $Q(l_j \to l_k, t) = Q_V(l_k, t)$, where subscript is set to V because Q_V is similar to state value function V. This is ignoring cost of moving l_j to l_k, but greatly reduces the complexity of problem. Then, we can write Bellman expectation equation as following.

$$V_{\pi_\phi}(l_j, t) = (1 - p_{j,t}^c) Q_{V,\pi_\phi}(l_j, t) + p_{j,t}^c \sum_{l_k \in S(l_j)} \pi_\phi(l_j \to l_k, t) Q_{V,\pi_\phi}(l_k, t) \tag{3}$$

$$Q_{V,\pi_\phi}(l_j, t) = \mathbb{E}[R_{j,t}^i | \text{agent } i \text{ is at } l_j \text{ at time } t \text{ after relocation}] + \gamma V_{\pi_\phi}(l_j, t+1) \tag{4}$$

where $p_{j,t}^c$ means probability to be controllable and $S(l_j)$ means successor road set of l_j.

4 Multi-agent reinforcement learning with graph neural network

4.1 Large scaled multi-agent reinforcement learning

Basically we will use Q learning to handle our MARL problem as [11, 26, 10]. However, there are few things to think about. In conventional RL, optimal policy π^* satisfies following greediness property where Q_* is an optimal action-value function.

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \operatorname*{argmax}_a Q_*(s, a) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

However, in large scale multi-agent problem, we cannot assume this optimal policy greediness. It may be valid if we update the state and policy by moving agents one by one. However it will cost a lot of computation, so we assume that we control all agents simultaneously. In this case, if all agents are sent to the road with the highest action-value, there will be a negative impact on the performance. This is intuitively clear considering simple case that allocating 10 drivers to two roads who have 3 and 7 calls each. Optimal policy for our problem setting should be stochastic, not all-or-nothing. Therefore, any assumption related to optimal policy greediness or Bellman optimality equation should be modified in our problem setting. For example, it is undesirable to use original Q value update in standard Q-learning as [11].

$$Q(s, a) \leftarrow Q(s, a) + (1 - \alpha)(R(s, a, s') + \gamma \max_{a'} Q(s', a')) \tag{6}$$

Instead we may use following update rule which is also known as *expected-SARSA* [18]:

$$\begin{aligned} Q(s,a) \leftarrow & Q(s,a) + (1-\alpha)\left(R(s,a,s') + \gamma \mathbb{E}[Q(s',a')]\right) \\ = & Q(s,a) + (1-\alpha)\left(R(s,a,s') + \gamma\left(\sum_{a'} \pi(a'|s')Q(s',a')\right)\right) \end{aligned} \tag{7}$$

Policy update rule should be also changed from greedy update. Unfortunately, it is a huge task to find an optimal stochastic policy update method. One plausible setting is to increase the probability of a whose Q value is larger. We can simply achieve this by following setting where F_π is an increasing function of Q.

$$\pi(a|s) = F_\pi(Q(s,a)) \tag{8}$$

There are countless options to choose F_π, but here, we simply set it to normalized β-squared function as follows. Note that if $\beta = \infty$, then it is same with greedy update.

$$\pi(a|s) = \frac{Q(s,a)^\beta}{\sum_a Q(s,a)^\beta} \tag{9}$$

Note that overall value/policy update is similar to *mean field* in [24]. However they used stochastic policy in perspective of exploration and set policy to become greedy at the end, which is different from our usage. Generally saying, there is no guarantee that our update method converges to an optimal solution. This can be easily shown with a simple counter-example. However, at least we can expect better result than using a greedy policy. We will discuss this more in supplementary material.

Now, let's rewrite Eq.(7) to fit with out problem setting. We consider updating single agent's experience whose action is $l_t^i \to l_{t+1}^i$.

$$Q_V(l_{t+1}^i, t) \leftarrow Q_V(l_{t+1}^i, t) + (1-\alpha)\left(R_t^i + \gamma T(l_{t+1}^i, t+1)\right) \tag{10}$$

where

$$T(l_j, t) = \begin{cases} \displaystyle\sum_{l_k \in S(l_j)} \pi(l_j \to l_k, t) Q_V(l_k, t), & \text{if agent is controllable} \\ Q_V(l_j, t), & \text{if agent is non-controllable} \end{cases} \tag{11}$$

Note that $\mathbb{E}[T(l_j,t)] = V(l_j,t)$. Meanwhile, Eq.(9) can be rewritten as follows:

$$\pi(l_j \to l_k, t) = \frac{Q_V(l_k,t)^\beta}{\sum_{l_k \in S(l_j)} Q_V(l_k,t)^\beta} \tag{12}$$

DQN [13], one of the most famous deep RL techniques, uses neural network to approximate Q value function in Q learning. We also adopt methodology of DQN, but have modified it slightly to suit the problem situation. Mean-square loss for i th agent at time t whose action is $l_t^i \to l_{t+1}^i$, is

$$\left[Q_V(l_{t+1}^i, t; \theta) - \left(R_t^i + \gamma T'(l_{t+1}^i, t; \theta')\right)\right]^2 \tag{13}$$

where $T'(l^i_{t+1}, t; \theta')$ refers to Eq.(11) using Q'_V, π', calculated from target neural network with parameter θ'. Our goal is to minimize above MSE loss of all agents. The overall training process is summarized in **Algorithm 1**. For simplicity, we set driver's state to done if it newly serves order. Also we do not use a replay memory, because the correlation of experiences from multiple agents is already diluted.

Algorithm 1 Modified DQN with stochastic policy update

```
1:  Initialize $Q$ with $\theta$
2:  Initialize $Q'$ with $\theta' = \theta$
3:  for m = 1 to maxiter do
4:      Reset environment and observe initial state $s_0$.
5:      for t = 0 to T do
6:          Compute $Q_V(l_j, t; \theta)$ from $s_t$
7:          Compute $\pi(l_j \to l_k, t; \theta)$ from $Q_V(l_j, t; \theta)$ using Eq.(12)
8:          Sample next action $a^i_t$ from $\pi(l^i_t, t; \theta)$
9:          Apply joint action $[a^i_t]^{N_t}_1$ and observe $[R^i_t]^{N_t}_1, s_{t+1}$
10:         Compute $Q'_V(l_j, t+1; \theta')$ from $s_{t+1}$
11:         Compute $\pi'(l_j \to l_k, t+1; \theta')$ from $Q'_V(l_j, t+1; \theta')$ using Eq.(12)
12:         for each idle driver $i$ do
13:             Set $y^i_t = \begin{cases} R^i_t, & \text{if get call} \\ \gamma T'(l^i_{t+1}, t+1; \theta'), & \text{otherwise} \end{cases}$
14:         end for
15:         Update $\theta$ by a gradient descent on loss function $\sum_{i=1}^{N_t} \left[ y^i_t - Q_V(l^i_{t+1}, t; \theta) \right]^2$
16:         Update target network parameters $\theta' \leftarrow \theta$ if needed.
17:     end for
18: end for
```

4.2 Graph neural network as a function-approximator

To approximate aforementioned Q_V, we use a graph neural network(GNN). GNN takes a graph $G = (V, E)$ and a d-dimensional graph signal $X \in \mathbb{R}^{|V| \times d}$ as an input. We use two basic models, Graph Convolutional Network(GCN) [7] and Graph Attention Network(GAT) [19]. For both GCN and GAT, a graph convolution operation of ith node at lth layer can be written as follows:

$$h_i^{(l+1)} = \sigma \left(\sum_{j \in \mathcal{N}(i)} c W^{(l)} h_j^{(l)} \right) \tag{14}$$

where $h_i^{(l)}$ is a lth layer node embedding, $\mathcal{N}(i)$ is a set of node i's neighbors, σ is a non-linear activation function such as ReLU or sigmoid, and $W^{(l)}$ is a learnable weight matrix for node-wise feature transformation. For GCN, c is set to be a normalizing constant $1/(\sqrt{|\mathcal{N}(i)|}\sqrt{|\mathcal{N}(j)|})$ which is dependent on graph structure. For GAT, c is set to be a normalized attention score $\alpha_{ij}^{(l)}$ which is calculated from additional learnable

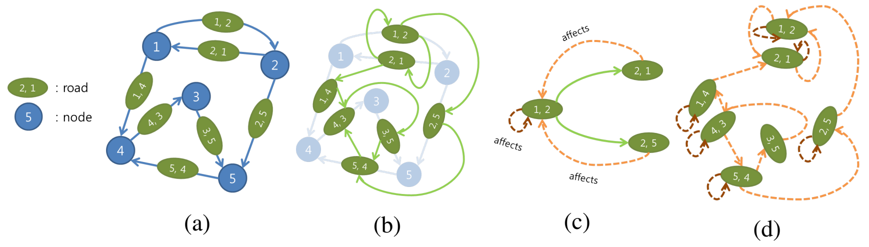


Figure 2: Converting road network G_R to a graph with proper form that is easy to apply graph neural networks. (a): original road network (b): line graph conversion (c): successor road and road itself affect the road's Q_V value. (d): final graph after conversion.

parameters. So far, the case of undirected graph has been described, but we can apply same method to directed graph by replacing neighbors to predecessor.

In this problem, we want to design GNN takes $s_t \in \mathbb{R}^{N_{road} \times 3}$ as an input and generates $Q_V(t) \in \mathbb{R}^{N_{road}}$ as an output. In order to properly do this, we have to transform the road network G_R as illustrated in Fig. 2. First, to handle edges as nodes, we convert G_R to its edge-vertex dual graph, or a line graph, $L(G_R)$ (Fig. 2(b)). Now think about the two components of Q_V update target: reward term R and discounted future return T. The reward term is affected by the road itself, while discounted future return is affected by its successors. Note that in GCN and GAT, the message is passed to successors, so $h_i^{(l)}$ is computed by its predecessors. Thus, in order to successor road affect predecessor road, we have to reverse the direction of edges. We also have to add self-loop. The final graph becomes like Fig. 2(d).

5 Simulator design

Unlike standard supervised learning problems which have fixed data set, reinforcement learning is hard to train and evaluate because of its interactive property. One common practice is to build a simulator that can reproduce the environment of the specific RL problem. Unfortunately, for many traffic studies which contain enormous amount of data, it is nearly impossible to fully reproduce the real traffic environment. Thus, designing manageable, but realistic simulator is a crucial factor in traffic studies. Here, we introduce a simulator design that can models real taxi-hailing service.

To run the simulator, we need call data within a period, the number of total drivers over time, and initial idle driver distribution. At the initialization, we deploy idle drivers at each road l_j corresponding to initial idle driver distribution, and assign calls. At each time step t, we go through following cycle taking RL model's policy π_t as an input and return new observation s_{t+1}.

1. Move every drivers forward by its current speed. If a driver can move further than the end of its current road, add it to controllable driver list.

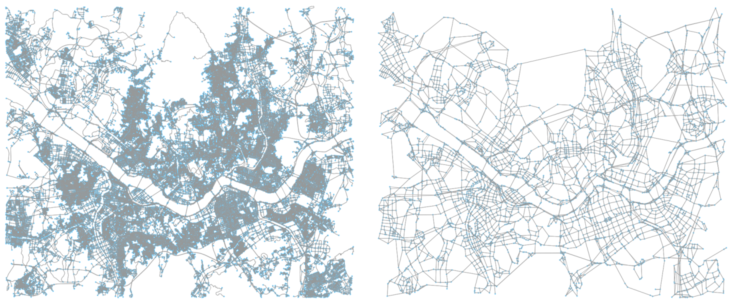


Figure 3: Original road network of Seoul which has 85,345 nodes and 243,621 edges(left) and the simplified road network which has 4,553 nodes and 13,334 edges(right).

2. Apply policy π_t to the controllable drivers and relocate them to the next road. Relative position on new road is set to random. Also update non-controllable drivers' position.

3. Assign orders to the drivers. Orders can be only assigned to the drivers at the same road.

4. Increase time step by 1 and check whether each driver has finished current job.

5. Generate orders at each road l_j using data $Call(j, t)$. Call data is formulated as (start road, end road, start time, duration, cost) pair.

6. To consider drivers go offline or online, generate or remove drivers to fit the total number of drivers $N_D^{total}(t)$. Also remove expired orders and set new speed to each road.

7. Observe next state $s_{t+1} = [(N_{j,t+1}, N_{j,t+1}^{call}, speed_{j,t+1})]_{j=1}^{N_{road}}$.

We design the simulator on two different environments. To find an optimal β in Eq.(9), we first run simulation on a simple 10 by 10 grid shaped city. The input data is all artificially generated. The simulator is then tested on a real road network in Seoul. Because the original network was too large, we simplified it through several steps. The required data are provided by company K. Speed data is gathered from public data sites. The details can be found in supplementary material.

6 Experiments

6.1 Experimental setting

We trained and evaluated on above two environments. For synthetic case, total number of time step of one episode is 100 and trained for 20 epochs, tested for 10 epochs. For real case, we split one day to 1440 steps(1 minute for each step), and trained for 5 epochs, tested for 2 epochs. We use 8 layers for both GCN and GAT. For GAT we also used 8 attention heads. We use ReLU as an activation function. We balanced exploration/exploitation by setting policy to $(1-\epsilon)\pi + \epsilon\pi_r$ where π is our calculated policy and π_r is a uniform distribution policy. ϵ is linearly annealed from 1 at the start and to 0 at the end of training. No exploration was made in test steps. Discount factor γ is set to 0.9.

6.2 Performance comparison

We evaluated several baseline methods and compare the performance with our models. For GCN-DQN and GAT-DQN, policy is calculated from Eq.(12).

- **Random**: This method randomly diffuses drivers to successor roads.
- **Proportional**: This method deploys drivers proportional to the number of orders.
- **GCN-DQN**: This method uses GCN to approximate Q_V values in DQN.
- **GAT-DQN**: This method uses GAT to approximate Q_V values in DQN.

In order to compare the performance of each method, we calculated order response rate, which is dividing number of served order by total orders.

The result for simple city is on Table 1. There was no significant difference for different βs, but the performance degradation was noticeable when $\beta = \infty$. Anyway, $\beta = 1.5$ gave the slightly better result than the others, so we used this value for simulating real scenario.

The result for real Seoul is on Table 2. To test the robustness of our algorithm, we tested our algorithm with different driver numbers. Here, $n\%$ driver means that reducing every driver quantity related values such as initial driver distribution, or total driver number at each time step t. We also tested simulation on other days. Case (A) is same day of the week, different date which has similar call/driver distribution. Case (B) is different day of the week, which has different call/driver distribution. GCN-DQN and GAT-DQN showed better result than simple rule-based method. Also, using greedy policy update($\beta = \infty$) was shown to be harmful for simultaneous multi-agent control, which is consistent with our intuition. It seems GCN is more vulnerable to this degradation. In addition, overall, GAT-DQN was better than GCN-DQN, which is thought to be because GAT is more expressive than GCN.

6.3 Qualitative result

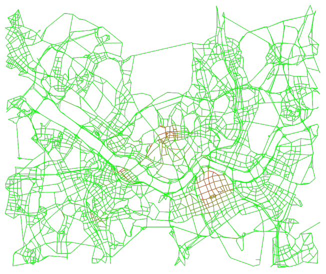
Finally we report the qualitative result of our work (Fig. 4). We plotted road network that shows Q_V of each road. At 1:00 AM, area nearby entertainment district showed higher Q_V values. While at 8:00 AM, high values were found throughout the city, which is due to commuting people. Overall, we could conclude that Q_V value calculated by our model reflects real situation well.

Table 1: Performance comparison using different β on Eq.(12) for simple synthetic road network.

Method	Order response rate						
	$\beta = 0.1$	$\beta = 0.5$	$\beta = 1.0$	$\beta = 1.5$	$\beta = 2.0$	$\beta = 3.0$	$\beta = \infty$
GCN-DQN	0.6827	0.6876	0.6878	0.6804	0.6840	0.6810	0.4029
GAT-DQN	0.8002	0.8050	0.8015	**0.8771**	0.7806	0.8070	0.6617

Table 2: Performance comparison using $\beta = 1.5, \infty$ on real road network of Seoul with different number of drivers. (A) means simulation on same day, different date. (B) means different day.

Method	Order response rate				
	20% driver	50% driver	100% driver	(A)	(B)
Random	0.2258	0.4613	0.6697	0.7092	0.7109
Proportional	0.2383	0.4900	0.6940	0.7247	0.7245
GCN-DQN$_{\beta=\infty}$	0.2218	0.4089	0.5672	0.5416	0.5413
GCN-DQN$_{\beta=1.5}$	0.2711	0.5475	0.7677	0.7756	0.7909
GAT-DQN$_{\beta=\infty}$	0.2930	0.5550	0.7497	0.7346	0.7400
GAT-DQN$_{\beta=1.5}$	**0.2980**	**0.5726**	**0.7771**	**0.7893**	**0.7973**


(a) 1:00 AM

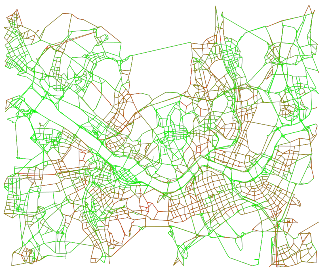

(b) 8:00 AM

Figure 4: Expected Q_V values of each road in Seoul. Red means higher Q_V value.

7 Conclusion

In this paper, we try to find an efficient fleet management strategy in ride-hailing service. We assume spatial condition as a graph which represents real world much better than a grid used in [11, 9, 26, 5]. We design proper Markov Game model for multi-agent RL on a graph. Modified DQN with stochastic policy update is adopted for efficient simultaneous multi agents control. We also show that GCN and GAT can effectively approximate the Q value of the graph. We develop a simulator that reflects reality, and use it as a training/testing environment. Under various conditions, we can demonstrate the effectiveness of our proposed framework.

Broader Impact

The ride-hailing and ride-sharing platforms have recently evolved significantly as the concept of vehicles moves from ownership to sharing. These services are an important factor in building a smart city in the future. However, it still suffers from optimization problem which degrades user experience and prevents users from using the service. In this study, we explored how to realistically model and optimize ride-hailing services. We expect that this study will improve the user experience of ride-hailing/ride-sharing services and encourage more people to use them.

References

[1] Aamena Alshamsi, Sherief Abdallah, and Iyad Rahwan. "Multiagent self-organization for a taxi dispatch system". In: *8th international conference on autonomous agents and multiagent systems*. 2009, pp. 21–28.

[2] Irwan Bello et al. "Neural combinatorial optimization with reinforcement learning". In: *arXiv preprint arXiv:1611.09940* (2016).

[3] Maxime Gasse et al. "Exact combinatorial optimization with graph convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 15554–15566.

[4] Gregory A Godfrey and Warren B Powell. "An adaptive dynamic programming algorithm for dynamic fleet management, I: Single period travel times". In: *Transportation Science* 36.1 (2002), pp. 21–39.

[5] Jintao Ke et al. "Optimizing Online Matching for Ride-Sourcing Services with Multi-Agent Deep Reinforcement Learning". In: *arXiv preprint arXiv:1902.06228* (2019).

[6] Elias Khalil et al. "Learning combinatorial optimization algorithms over graphs". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6348–6358.

[7] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[8] Der-Horng Lee et al. "Taxi dispatch system based on current demands and real-time traffic conditions". In: *Transportation Research Record* 1882.1 (2004), pp. 193–200.

[9] Minne Li et al. "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning". In: *The World Wide Web Conference*. 2019, pp. 983–994.

[10] Xihan Li et al. "A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2019, pp. 980–988.

[11] Kaixiang Lin et al. "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*. ACM. 2018, pp. 1774–1783.

[12] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[13] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[14] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications". In: *IEEE transactions on cybernetics* (2020).

[15] Afshin OroojlooyJadid and Davood Hajinezhad. "A review of cooperative multi-agent deep reinforcement learning". In: *arXiv preprint arXiv:1908.03963* (2019).

[16] Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. "A collaborative multiagent taxi-dispatch system". In: *IEEE Transactions on Automation Science and Engineering* 7.3 (2009), pp. 607–616.

[17] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[18] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.

[19] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[20] Xiaoqiang Wang et al. "Large-scale Traffic Signal Control Using a Novel Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1908.03761* (2019).

[21] Zhaodong Wang et al. "Deep reinforcement learning with knowledge transfer for online rides order dispatching". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 617–626.

[22] Chong Wei et al. "Look-ahead insertion policy for a shared-taxi system based on reinforcement learning". In: *IEEE Access* 6 (2017), pp. 5716–5726.

[23] Zhe Xu et al. “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*. ACM. 2018, pp. 905–913.

[24] Yaodong Yang et al. “Mean field multi-agent reinforcement learning”. In: *arXiv preprint arXiv:1802.05438* (2018).

[25] Lingyu Zhang et al. “A taxi order dispatch model based on combinatorial optimization”. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 2151–2159.

[26] Ming Zhou et al. “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*. 2019, pp. 2645–2653.