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

Juhyeon Kim[1,2,†] and Kihyun Kim[2]

***Abstract*— We propose a novel approach to optimize fleet management by combining multi-agent reinforcement learning with graph neural network. To provide ride-hailing service, one needs to optimize dynamic resources and demands over spatial domain. While the spatial structure was previously approximated with a regular grid, our approach represents the road network with a graph, which better reflects the underlying geometric structure. Dynamic resource allocation is formulated as multi-agent reinforcement learning, whose action-value function (Q function) is approximated with graph neural networks. We use stochastic policy update rule over the graph with deep Q-networks (DQN), and achieve superior results over the greedy policy update. We design a realistic simulator that emulates the empirical taxi call data, and confirm the effectiveness of the proposed model under various conditions.**

I. Introduction

One of the main challenges in ride-hailing services is supply and demand balancing—making vehicles available at the right place and time for customers. To achieve this goal, establishing a smart and timely reallocation strategy for dynamic transportation resources is crucial, which is known as a *fleet management problem*. The goal of this optimization problem is to maximize the number of matches between customers and drivers by appropriately directing idle drivers to proper regions. Without such efforts, customers will suffer from the lack of drivers, while some drivers will remain idle in near areas looking for orders. Though there exist rich empirical demand and supply data, designing an optimal vehicle management system still remains a challenging task. One of the main obstacles is the dynamic nature of the fleet management problem that the current relocation policy affects the future demand-supply distribution.

To resolve such difficulty, reinforcement learning (RL) has been widely adopted in previous works [1], [2]. In RL frameworks, agents iteratively learn a strategy to maximize the reward by interacting with an environment. To establish a strategy that has practical implications, an accurate but manageable problem setting and RL framework design is critical. To this end, recent studies [3]–[6] suggested multi-agent deep reinforcement learning (MADRL) to effectively approximate highly complicated fleet management scenario. Though these works have brought an important insight, they still have limitations to be applied in real systems since they oversimplified the spatial structure by assuming a grid-shaped environment. If a graph is used to model the environment, it would be possible to allocate drivers more precisely at the road level. Yet using a graph with previously proposed approaches is difficult since a graph requires a relatively complex structure whose components are connected in an irregular manner.

This paper proposes a novel approach to a fleet management problem using MADRL with graph neural network (GNN). Our study is based on the aforementioned papers [3]–[6], while we modify their settings to fit with our graph model of a road network. Specifically, while drivers in the grid-based approaches should only move to one of the neighboring grid cells, drivers in our model can move to any connected roads. Moreover, the exact location of each driver can be pinpointed in our model using the relative position on its current road, which was impossible in the previous works. As a result, our model can handle each order and driver more precisely at the road level. To find an effective relocation policy, we suggest a novel method using GNN to precisely estimate an action-value function (i.e. Q function). Inspired by the fact that the Q function of each road is highly dependent on its connected roads, we transform an entire road network to the appropriate GNN model and train this model using DQN. Finally, we suggest a simple stochastic policy update from Q function to efficiently handle a large number of drivers. Our observation shows that employing a stochastic policy improves the overall performance compared to conventional ϵ-greedy policy, by reasonably distributing multiple drivers to the next road according to the Q function.

Our major contributions are listed as follows:

- We propose a graph-based MADRL setting for a fleet management problem, which enables a more precise representation of a real-world scenario than the previous grid-based approaches.
- We specify a proper Markov game model and propose an efficient learning strategy to handle a large number of agents using a stochastic policy update. We also show that GNN can be used to approximate Q function on our graph model.
- We design a simulator to compare several algorithms using empirical data from *Kakao Mobility*. Using our simulator, we observe that our proposed method shows better performance than existing approaches.

II. Related Work

A. Large-scale fleet management

A fleet management problem has been extensively studied in a rule-based manner. For example, there exist works using

[1]J. Kim is with Kakao Mobility, Seongnam 13529, Republic of Korea
[2]J. Kim and K. Kim are with the Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, Republic of Korea {cjdeka3123, hahakhkim}@snu.ac.kr
[†]Corresponding author

greedy matching [7], centralized combinatorial optimization [8], collaborative dispatch with decentralized setting [9], or adaptive multi-agent scheduling system [10]. To exclude heuristics in the rule-based model, [11] and [12] adopted an RL-based approach. However, they assumed a single-agent setting that cannot model the complex interactions between large numbers of drivers and orders.

The more recent trend in fleet management studies is the adoption of MADRL. [3] proposed a contextual MADRL that considers geographic and collaborative contexts to prevent invalid actions. A cooperative MADRL algorithm is suggested by [4], which adopted mean-field approximation to model agents' interaction. [5] proposed a decentralized system based on MADRL with Kullback–Leibler divergence optimization. Besides, [6] adopted delayed order matching with a two-stage framework utilizing both MADRL and combinatorial method.

However, these studies assumed a grid world system that has highly different geometrical structures from real roads. Though there exist several studies dealing with graph combinatorial optimization with RL [13], [14], or without RL [15], a large-scale problem such as fleet management on a graph structure using MADRL has been rarely studied.

B. Multi-agent reinforcement learning

In multi-agent reinforcement learning (MARL) frameworks, a large number of agents interact with the same environment and receive rewards corresponding to their state and action. [16] is one of the first papers that dealt with a multi-agent scheme, proposing a cooperative Q-learning that agents share their policy and experience. [17] used counterfactual multi-agent policy gradient model which comprised of centralized critic and decentralized actor. [18] also used similar approach to [17], adding extra information in training phase. However, these works are limited to a handful number of agents due to communication costs. [19] alleviated this complexity by a homogeneity assumption and introduced a highly scalable multi-agent simulation platform called MAgent. [20] further developed MAgent with *mean-field* theory that approximates interactions between agents, and experimentally showed that it converges to the Nash equilibrium.

C. Reinforcement learning with stochastic policy

In several decision-making problems including a large-scale fleet management problem, an optimal policy cannot be modeled by a greedy (all-or-nothing) policy—sending all drivers in the same direction would obviously be detrimental. This is not a critical issue in policy-based RL, such as actor-critic or A3C [21], because there exists an additional network for approximating the stochastic policy. In value-based RL, the most popular approach that introduced the stochasticity is soft Q-learning [22]. Soft Q-learning expresses an optimal policy with a Boltzmann distribution by adding an entropy term. However, we found that the additional entropy term is useless and even harmful in the fleet management problem since drivers get direct negative feedback by over-concentration. Instead, we propose a simple stochastic policy using the function of Q values and experimentally show that this approximation is more effective than the former one.

III. Problem Statement

In this section, we specify a problem setting of the fleet management problem. First of all, we use the total order response rate as an objective to be maximized, instead of using the gross merchandise volume (GMV, the price sum of all served orders) that has been used in previous works [3], [4], [11]. The order response rate is the number of served orders divided by the number of total orders. The advantage of using the order response rate over GMV is that it can reflect the customers' satisfaction as well as the revenue. Other problem settings and notations are similar to those of previous works [3], [4], [6], [11], [23]. The major difference is a spatial setting—we use a graph rather than a grid. For a temporal setting, we equally divide one day to T time steps.

A. Graph representation of a road network

A road network can be modeled as a directed graph $G_R := (V, E)$, where V denotes the set of intersections and $E := \{l_j \mid j = 1, 2, \ldots, N_{\text{road}}\}$ denotes the set of roads. All roads are assumed directional, so even adjacent lanes between the same nodes 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). The location of any driver can be represented by the road index and the relative position on the road.

At each time step, all drivers move forward by the speed of their current road. Staying at the current position is not allowed in our setting. If a driver cannot reach the end of the current road, thereby it cannot transit to a different road within the current time step, we consider it as a *non-controllable* driver. Conversely, if a driver can reach the end of the current road, it is regarded as a *controllable* driver who can move on to the next roads. Each controllable driver should be relocated to one of its connected roads. In this problem, we randomly set the relative position of relocated agent on the new road, according to the uniform distribution. It reflects the time spent at the previous intersection and also gives randomness to our environment. One may concern that giving this randomness greatly affects the ratio of controllable/non-controllable agents, but we observed that this ratio is similar to the deterministic setting. We also limit the number of transitions to 1 time per time step for any driver. Instead, we shorten the unit time to make the movable distance per time step short enough.

Orders (calls) are generated from each road at each time step and are randomly assigned to idle drivers at the same road. Here, idle drivers are defined as those who do not currently serve any order. For simplicity of the problem, we do not consider the order's relative position on the road. Under this setting, our ultimate goal is to find an optimal relocation strategy for controllable drivers that maximizes the order response rate.

B. Markov game modeling

Now we define a Markov game $G := (N, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. $(N, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ refers to the total agent number, state

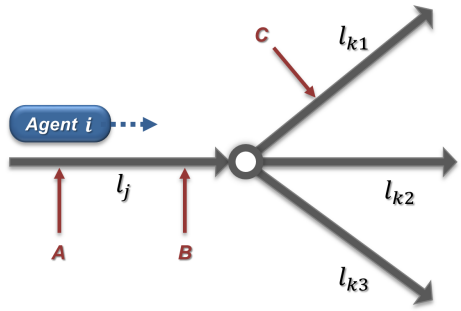


Fig. 1: Illustration of a road network. The agent i is currently located at the position A on the road l_j. At the next time step, it moves forward. Depending on the speed of the road l_j, it can either stay on the same road (position B) or move on to one of the neighboring roads l_ks (position C).

space, joint action space, transition probability, reward function, and discount factor, respectively.

- Agent : We define an agent as a driver who is at the idle state (currently not serving an order). As mentioned above, an agent can be either controllable or non-controllable. Controllable agents at the same road and the time are assumed to be homogeneous, implying that they share the same policy. Orders are randomly assigned to both types of agents regardless of their relative position on the road. Let N_t denote the number total agents at time t. We denote the road that ith agent is located on at time t as l_t^i, where $i = 1, 2, \ldots, N_t$. We also define A_t^c as the set of all controllable agents at time t.
- State $s_t \in \mathcal{S}$: We use the global state s_t at each time t. We observe three values at each road l_j at time t: the number of agents (idle drivers) $N_{j,t}$, the number of calls $N_{j,t}^{\text{call}}$, and the speed $speed_{j,t}$. We concatenate this information for all roads to obtain the global state
$$s_t := [(N_{j,t}, N_{j,t}^{\text{call}}, speed_{j,t})]_{j=1}^{N_{\text{road}}} \in \mathbb{N}_0^{N_{\text{road}} \times 2} \times \mathbb{R}^{N_{\text{road}}}. \tag{1}$$
s_t will be used as an input for a graph neural network.
- Action $a_t \in \mathcal{A}$: As mentioned above, all controllable agents must decide which road to move on. We define the action a_t^i as the decision of the next road for the agent i at time t. Note that each road has a different set of possible actions. For simplicity, we will indicate the action moving from l_j to l_k as $l_j \to l_k$. For non-controllable agents, there is no freedom to select an action. In other words, their action is limited to stay on the current road (e.g. $l_j \to l_j$). We aggregate all agents' action to define a joint action $a_t := [a_t^i]_{i=1}^{N_t}$.
- Reward $\mathcal{R}_t \in \mathcal{R}$: $\mathcal{R}_t := [\mathcal{R}_t^i]_{i=1}^{N_t}$, where $\mathcal{R}_t^i$ is defined as the ith agent's reward at time t. After the action a_t^i, R_t^i is simply set to 1 if an order is assigned to the ith agent and 0 if not. Note that non-controllable drivers can also serve calls and receive the reward.
- State transition probability $\mathcal{P}$: Recall that the global state includes three different types of information. The number of idle agents ($N_{j,t}$) depends on the previous state and action, while the number of orders and the speed of each road are given by fixed data. We also add or remove idle drivers on each road according to the normal distribution, because the number of drivers varies in the real world. The relative position of newly added drivers on each road is determined according to the uniform distribution.

The ith agent's discounted return at time t is given by $G_t^i := \sum_{k=0}^{\infty} \gamma^k R_{t+k}^i$, where $\gamma \in (0, 1)$ is a discount factor. To consider the road-level policy, we define $\pi(l_j \to l_k | s_t)$, which denotes the probability of the action $l_j \to l_k$ at the road l_j. By definition, $\sum_{l_k \in S(l_j)} \pi(l_j \to l_k | s_t) = 1$ for $\forall s_t \in \mathcal{S}$, $\forall j = 1, 2, \ldots, N_{\text{road}}$, where $S(l_j)$ denotes the set of all successor roads of l_j.

C. Bellman expectation equation

We define the state-value function and the action-value function (Q function) as follows:

$$\begin{aligned} V^\pi(s_t, l_j) &:= \mathbb{E}^\pi[G_t^i \mid \text{agent } i \text{ is on } l_j \text{ at time } t \text{ before} \\ &\qquad \text{movement, under the state } s_t], \\ Q^\pi(s_t, l_j \to l_k) &:= \mathbb{E}^\pi[G_t^i \mid \text{agent } i \text{ moves } l_j \to l_k \\ &\qquad \text{at time } t, \text{ under the state } s_t]. \end{aligned} \tag{2}$$

Here, $\mathbb{E}^\pi$ denotes the expectation value when all agents move according to the policy π. Note that our state-value function $V^\pi(s_t, l_j)$ is marginalized over the controllability of all agents on the road l_j. In other words, $V^\pi(s_t, l_j)$ is an expected value for both controllable and non-controllable agents.

To simplify our formulation, we need two assumptions. First, we assume that the expected reward does not depend on the agent's relative position on the road. Thus, the expected reward of any agent on the road l_j is identical as $V^\pi(s_t, l_j)$ regardless of its relative position on l_j. Second, we assume that $Q^\pi(s_t, l_j \to l_k)$ does not depend on the departed road l_j (as in [3]), which allows us the following approximation:

$$Q^\pi(s_t, l_j \to l_k) \approx Q^\pi(s_t, l_k), \quad \forall j = 1, 2, \ldots, N_{\text{road}}. \tag{3}$$

Though we cannot take account of the cost of moving from l_j to l_k under this assumption, we can greatly reduce the complexity of problem. Remark that approximated $Q^\pi(s_t, l_k)$ has the same input-output structure to the state-value function $V^\pi(s_t, l_j)$.

The state-value function can be expressed as follows by dividing into non-controllable case and controllable case:

$$\begin{aligned} V^\pi(s_t, l_j) \approx{}& (1 - p_{j,t}^c) Q^\pi(s_t, l_j) \\ &+ p_{j,t}^c \sum_{l_k \in S(l_j)} \pi(l_j \to l_k | s_t) Q^\pi(s_t, l_k), \end{aligned} \tag{4}$$

where $p_{j,t}^c$ denotes the probability that an agent on the road l_j at the time step t is controllable. The Bellman expectation equation is then given by

$$\begin{aligned} Q^\pi(s_t, l_k) = \mathbb{E}[R_t^i \mid{}& \text{agent } i \text{ is on } l_k \text{ at time } t \text{ after} \\ &\text{movement, under the state } s_t] + \gamma V^\pi(s_{t+1}, l_k). \end{aligned} \tag{5}$$

(4) and (5) will be used to estimate Q function in the next section.

IV. Multi-agent Reinforcement Learning with Graph Neural Network

A. Large-scale multi-agent reinforcement learning

We use Q-learning to estimate the Q function in our MARL problem as in [3], [5], [24]. In conventional RL, an optimal policy π^* satisfies the following greediness property, where Q_* is an optimal action-value function:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} Q_*(s,a) \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

However, in a large-scale multi-agent problem, this greediness property does not hold. It may be valid if we can update the state and policy by moving each agent one by one. Nevertheless, because of the computational cost, our problem setting assumes that all agents must be controlled simultaneously. Therefore, if we employ a greedy policy, all agents on the same road will be sent to the same next road according to the highest action-value function. However, it is intuitively clear that sending all agents to the same next road will not be optimal in most cases.

As a result, an optimal policy for our problem setting would be stochastic (randomized), not all-or-nothing. [3] dealt with this by simply keeping a small portion of randomness with the ϵ-greedy policy ($\epsilon = 0.1$) during evaluation. However, this approach has a limitation that only the maximum Q-value is considered in the stochastic policy. Instead, we propose a method that does not rely on the optimal policy greediness or the Bellman optimality equation. We replace Q function update in the standard Q-learning to the following update rule, which is known as *expected-SARSA* [25]:

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

where $\mathbb{E}[Q(s',a')] = \sum_{a' \in \mathcal{A}} \pi(a'|s') Q(s',a')$. A policy update rule is also modified in our method. Unfortunately, it is difficult to find an optimal stochastic policy update method in a complex model such as a fleet management problem. One plausible setting is to increase the probability of the action which has a large Q value. We can simply achieve this by setting $\pi(a|s) = F_\pi(Q(s,a))$, where F_π is an increasing function of $Q(s,a)$ satisfying $\sum_{a \in \mathcal{A}} F_\pi(Q(s,a)) = 1$ for $\forall s \in \mathcal{S}$. There are countless options for F_π, but here, we simply set it to normalized β-squared function or β-exponential function as follows:

$$\pi(a|s) = \frac{Q(s,a)^\beta}{\sum_{a \in \mathcal{A}} Q(s,a)^\beta} \text{ or } \frac{\exp(\beta Q(s,a))}{\sum_{a \in \mathcal{A}} \exp(\beta Q(s,a))}. \tag{8}$$

Note that these stochastic policies converge to a greedy policy as $\beta \to \infty$.

Overall value/policy update is similar to that of maximum entropy based approaches or soft Q-learning [22]. One major difference is the presence of an entropy term. Soft Q-learning encouraged the stochastic behavior by introducing an additional entropy term. However in our problem setting, since the greedy policy leads to the immediate reduction of reward, there is no reason to add an entropy term for exploration. *Mean-field* in [4] is also similar to our approach, but they used the stochastic policy in the perspective of exploration and set their policy to become greedy at the end of the learning, which is different from our usage. Generally saying, there is no guarantee that our update method converges to an optimal policy. Nevertheless, we can expect a better result than using a greedy policy (see Appendix I for more discussion).

Now, we reformulate Q function update rule (7) with our problem settings and notations. Consider updating ith agent's experience whose state transition sample is given by $(s_t, a_t^i = l_t^i \to l_{t+1}^i, R_t^i, s_{t+1})$. From (4) and (5), our update rule can be expressed as

$$\begin{aligned} Q^\pi(s_t, l_{t+1}^i) \leftarrow &(1-\alpha) Q^\pi(s_t, l_{t+1}^i) \\ &+ \alpha \left[R_t^i + \gamma \hat{Q}^\pi(s_{t+1}, l_{t+1}^i, i) \right], \end{aligned} \tag{9}$$

where

$$\begin{aligned} &\hat{Q}^\pi(s_t, l_j, i) \\ &:= \begin{cases} \sum_{l_k \in S(l_j)} \pi(l_j \to l_k | s_t) Q^\pi(s_t, l_k) & \text{if } i \in A_t^c \\ Q^\pi(s_t, l_j) & \text{if } i \notin A_t^c. \end{cases} \end{aligned} \tag{10}$$

In (10), the case $i \in A_t^c$ denotes the expected future value of Q function when the agent i is controllable, and the case $i \notin A_t^c$ denotes the value when i is non-controllable. Remark that $\mathbb{E}[\hat{Q}^\pi(s_t, l_j, i)] = V^\pi(s_t, l_j)$ since $V^\pi(s_t, l_j)$ is marginalized over the controllability by definition. Our stochastic policy can be rewritten as follows using (8):

$$\begin{aligned} \pi(l_j \to l_k | s_t) = &\frac{Q^\pi(s_t, l_k)^\beta}{\sum_{l_k \in S(l_j)} Q^\pi(s_t, l_k)^\beta} \\ \text{or } &\frac{\exp(\beta Q^\pi(s_t, l_k))}{\sum_{l_k \in S(l_j)} \exp(\beta Q^\pi(s_t, l_k))}. \end{aligned} \tag{11}$$

DQN [26], one of the most famous DRL techniques, uses neural networks to approximate Q function in Q-learning algorithm. We adopt the methodology of DQN but modify it slightly to suit our problem setting. The mean squared error (MSE) for ith agent at time t is given by

$$\left[Q^\pi(s_t, l_{t+1}^i; \theta) - \left\{ R_t^i + \gamma \hat{Q}^{\pi'}(s_{t+1}, l_{t+1}^i, i; \theta') \right\} \right]^2, \tag{12}$$

where $\hat{Q}^{\pi'}(s_{t+1}, l_{t+1}^i, i; \theta')$ is defined as (10) using Q', π' calculated from the target neural network with parameter θ'. Our goal is to minimize the above MSE loss of all agents. The overall training process is summarized in **Algorithm 1**. For simplicity, we set the driver's state to be terminated if it newly serves an order. We do not use a replay memory, because the correlation of experiences from multiple agents is already diluted.

B. Graph neural network as a function-approximator

To approximate the Q function, 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

Algorithm 1 Modified DQN with stochastic policy update

Initialize Q with θ
Initialize Q' with $\theta' = \theta$
for $m = 1$ to maxiter **do**
  Reset environment and observe initial state s_0.
  for $t = 0$ to T **do**
    Calc $Q^{\pi}(s_t, l_j; \theta)$ from s_t
    Calc $\pi(l_j \rightarrow l_k | s_t; \theta)$ from $Q^{\pi}(s_t, l_j; \theta)$
    Sample next action a_t from π
    Apply action a_t and observe R_t, s_{t+1}
    Calc $Q^{\pi'}(s_{t+1}, l_j; \theta')$ from s_{t+1}
    Calc $\pi'(l_j \rightarrow l_k | s_{t+1}; \theta')$ from $Q^{\pi'}(s_{t+1}, l_j; \theta')$
    for each idle driver $i = 1$ to N_t **do**
      Set $y_t^i = \begin{cases} R_t^i \ (=1), & \text{if get call} \\ \gamma \hat{Q}^{\pi'}(s_{t+1}, l_{t+1}^i, i; \theta'), & \text{otherwise} \end{cases}$
    end for
    Update θ by a gradient descent on loss function $\sum_{i=1}^{N_t} [y_t^i - Q^{\pi}(s_t, l_{t+1}^i; \theta)]^2$
    Update $\theta' \leftarrow \theta$ if needed
  end for
end for

two basic models, GCN [27] and GAT [28]. For both GCN and GAT, a graph convolution operation of ith node at lth layer can be expressed as follows:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} c W^{(l)} h_j^{(l)} \right), \tag{13}$$

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 (e.g. ReLU, sigmoid), and $W^{(l)}$ is a learnable weight matrix for node-wise feature transformation. For GCN, c is set to be a normalizing constant $1/(\sqrt{|\mathcal{N}(i)|}\sqrt{|\mathcal{N}(j)|})$ that depends on the graph structure. For GAT, c is set to be a normalized attention score $\alpha_{ij}^{(l)}$ which is calculated from additional learnable parameters. So far, the case of an undirected graph has been described, but we can apply the same operation to a directed graph by replacing $\mathcal{N}(i)$ to a set of predecessor nodes.

We aim to design GNN that takes $s_t \in \mathbb{R}^{N_{\text{road}} \times 3}$ as an input and generates $Q(s_t) \in \mathbb{R}^{N_{\text{road}}}$ as an output. To this end, we transform the road network G_R as illustrated in Fig. 2. First, we convert G_R to its edge-vertex dual graph, or a line graph, $L(G_R)$ (Fig. 2(b)), because each road must be considered as a node. Now consider the two components of the update target in Q function: reward term R and discounted future return $\hat{Q}$. The reward term is affected by the current road, while the discounted future return is affected by its successors. Note that in both GCN and GAT, the message is passed to successors, meaning that $h_i^{(l)}$ is computed by its predecessors. Thus, in order to make successor roads affect predecessor roads, we need to reverse the direction of all edges (Fig. 2(c)). We also need to add a self-loop for each node to include the self-dependency required for computing the reward term. Fig. 2(d) illustrates the final shape of the graph model generated from the road network.

V. Simulator Design

Unlike standard supervised learning with fixed data set, RL is challenging to train and evaluate because of its interactive property. One challenging factor is building a simulator that reproduces the environment of the specific RL problem. Unfortunately, it is nearly impossible to fully reproduce real traffic environments due to the enormous amount of data. Thus, designing a manageable, but realistic simulator is a crucial factor in traffic studies. Here, we introduce a simulator that models real ride-hailing services.

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 step, we deploy idle drivers at each road l_j corresponding to the initial idle driver distribution and then assign calls. At each time step t, we go through the following cycle taking the policy π_t as input and returning new observation s_{t+1}.

1) Move every driver forward by the current speed of each road. If one can move further than the end of its current road, add it to the controllable driver list.
2) Relocate all controllable drivers to the next road by applying policy π_t. Relative position on the new road is set to random. Update non-controllable drivers' position.
3) Assign orders to the drivers. Orders can only be assigned to the drivers on the same road.
4) Increase time step by 1 and check whether each driver has finished current job or not.
5) Generate orders at each road l_j using the data $Call(j, t)$. Each call data is given by a tuple of (start road, end road, start time, duration, cost).
6) Generate or remove drivers to fit the given number of total drivers N_t^{total}, to consider drivers being offline or online. In addition, remove expired orders and set the new speed for each road.
7) Observe the next state

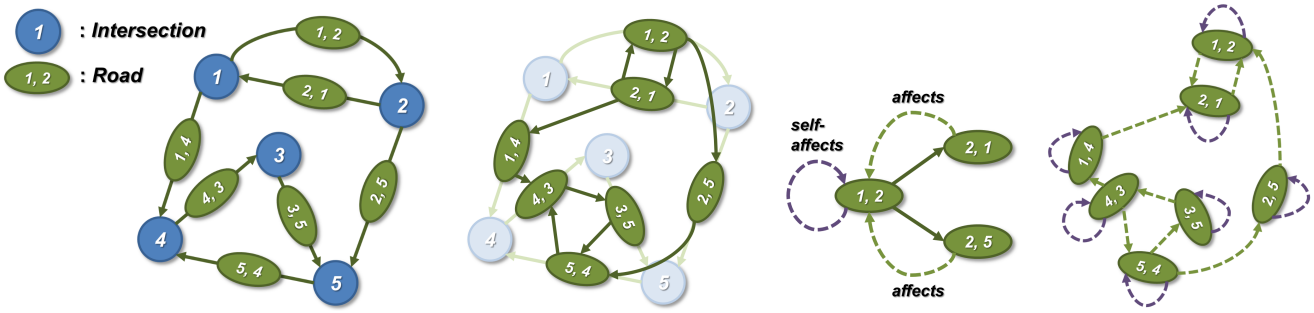
$$s_{t+1} = [(N_{j,t+1}, N_{j,t+1}^{\text{call}}, speed_{j,t+1})]_{j=1}^{N_{\text{road}}}.$$

We design the simulator on a real road network in Seoul, South Korea. Since the ground truth road network of Seoul is too large, we simplify it through several steps (Fig. 3).

VI. Experiments

A. Experimental setting

In this section, the performance of the proposed models is demonstrated and compared under various conditions. We trained and evaluated our algorithm using real taxi data in Seoul, provided by *Kakao Mobility*. The speed data of each road is gathered from the public website. We split one day into 1440 steps (1 minute for each step), trained for 5 epochs, and tested for 5 epochs. We used 8 layers for both GCN and GAT. For GAT, we used 8 attention heads. ReLU is used as an activation function except the last layer that used sigmoid to guarantee Q value between 0 and 1. We balanced exploration/exploitation by setting policy to $(1-\epsilon)\pi + \epsilon\pi_r$, where π denotes our computed policy, and π_r denotes a



(a) (b) (c) (d)

Fig. 2: Converting road network G_R to a proper graph form to apply graph neural networks: (a) original road network, (b) line graph conversion, (c) adding self-dependency and reversing direction, (d) the final graph after conversion.

Fig. 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).

policy using the uniform distribution for the action space. ϵ is linearly annealed from 1 at the start and to 0 at the end of the training. No exploration was made in the test steps since π is already stochastic. A discount factor γ is set to 0.9.

We evaluated several baseline methods and compared the performance with our methods.

- **Random**: This method randomly diffuses drivers to successor roads.
- **Proportional**: This method deploys drivers with the probability proportional to the number of orders in their successor roads.
- **DRL-based**: This method uses GCN or GAT to approximate Q function in DQN.

For DRL-based methods, the following strategies were compared.

- **ϵ-Greedy** [3]: This method sends all drivers to the next road that has maximum Q value, with the probability $1-\epsilon$, otherwise randomly diffuses drivers with the probability ϵ.
- **Entropy** [22]: This method is similar to **Exp**, whereas it uses soft value function Q^{soft} instead of Q as follows:

$$Q^{\text{soft}}(s,a) = R(s,a,s') + \frac{\gamma}{\beta}\log\sum_{a'\in\mathcal{A}}\exp\left(\beta Q^{\text{soft}}(s',a')\right), \quad (14)$$

which employs an entropy term to enforce exploration.

TABLE I: Performance comparison under various conditions. $n\%$ means number of drivers. (A) shows result on the same day, different date. (B) shows result on the different day.

Method	Order response rate				
	100%	50%	20%	(A)	(B)
Random	0.669	0.462	0.225	0.693	0.645
Proportional	0.694	0.490	0.238	0.708	0.665
ϵ-Greedy ($\epsilon = 0.0$)	0.685	0.505	0.269	0.646	0.626
ϵ-Greedy ($\epsilon = 0.1$) [3]	0.752	0.541	0.280	0.741	0.718
Entropy ($\beta = 20$) [22]	0.783	0.588	0.304	0.772	0.754
Pow ($\beta = 3$) (Ours)	**0.800**	**0.601**	**0.310**	**0.782**	**0.771**
Exp ($\beta = 20$) (Ours)	0.791	0.592	0.305	0.779	0.765

- **Pow / Exp**: This method sends drivers proportional to Q^{β} (power) or $\exp(\beta Q)$ (exponential) values in (11).

B. Results

Table I shows the order response rate of each proposed method under various conditions. Total 7 different methods suggested in Section VI-A are compared. Note that GAT is used for all of DRL-based methods. To analyze the robustness of our algorithm, we tested with the different number of drivers as in [3]. Here, $n\%$ driver ($n = 100, 50, 20$) means that we reduced the quantity of drivers including the initial driver distribution and the total driver number at each time step t. We also conducted simulations on different days, since call/driver distribution is highly dependent on the day of the week. Case (A) shows the result for the same day of the week (with a different date), which has a similar call/driver distribution. Case (B) shows the result for the different day of the week, which has different call/driver distribution.

In most cases, random dispatch showed the worst result as expected. The method dispatching proportional to the number of calls was slightly better than the random method. Overall, RL based methods were shown to outperform non-RL based methods. A greedy policy update ($\epsilon = 0$, $\beta = \infty$) performed worse than a stochastic policy update, which is consistent with our discussion in Section IV. Adding a small portion of randomness (i.e. $\epsilon = 0.1$) as in [3] gave better results but its performance was inferior to **Entropy**. Our proposed method that deploys agents proportional to the function of Q (**Pow**, **Exp**) showed the best performance. We could not

TABLE II: Performance comparison of GCN vs. GAT and **Exp** vs. **Entropy**. **Exp** was used for comparing GCN and GAT.

	100% driver		50% driver		20% driver	
β	GCN	GAT	GCN	GAT	GCN	GAT
50	0.771	0.776	0.537	0.575	0.262	0.297
20	0.778	0.791	0.546	0.592	0.267	0.305
10	0.775	0.782	0.545	0.583	0.267	0.300
5	0.773	0.775	0.546	0.578	0.268	0.300

	100% driver		50% driver		20% driver	
β	Exp	Entropy	Exp	Entropy	Exp	Entropy
50	0.776	0.780	0.575	0.580	0.297	0.298
20	0.791	0.783	0.592	0.588	0.305	0.304
10	0.782	0.759	0.583	0.563	0.300	0.293
5	0.775	0.657	0.578	0.465	0.300	0.256

find significant difference between **Pow** and **Exp**, but **Pow** gave slightly better result. Comparing the models used for GNN, GAT performed better than GCN which seems to be because GAT is more expressive than GCN (Table II).

The comparison of **Exp** and **Entropy** given in Table II provides us interesting insight. They use the same policy update rule (softmax on Q function), but **Entropy** method incorporates an entropy term to Q function to encourage exploration. The experimental result supports our argument that sending too many agents to the same road leads to instant negative feedback on the reward and makes entropy-based exploration superfluous or even detrimental. Compared to **Exp**, **Entropy** showed much dramatic performance degradation as β decreases. We observed that **Entropy** performed slightly better than **Exp** when β has a large value. This is because we cannot take advantage of the stochastic policy when β is too large, and forced exploration induced by the entropy term becomes somewhat helpful.

Finally, we report the qualitative result obtained from our work (Fig. 4). We plotted the road network that shows the value of Q function for each road. At 1:00 AM, areas near entertainment districts showed higher Q values. At 8:00 AM, high values were found throughout the city due to commuting people. Overall, we could conclude that Q values computed from our model reflect the real situation.

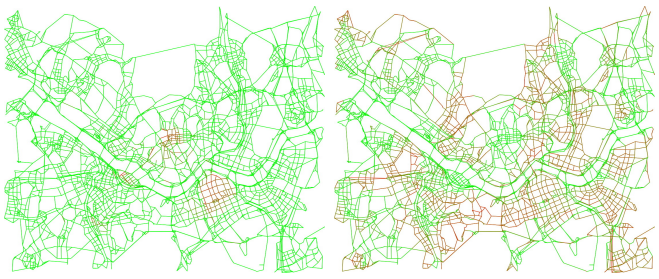


Fig. 4: Expected Q values of each road in Seoul at 1:00 AM (left) and 8:00 AM (right). A road with more red color has higher Q value.

VII. Conclusions

In this paper, we have presented a novel fleet management strategy in ride-hailing services. Our approach is distinguished from others by assuming a graph-based spatial condition, which has stronger representative power for road networks than a grid-based condition. Modified DQN with stochastic policy update is adopted and we showed that GNN can effectively approximate the Q function in our method. A simulator that reflects real road networks is designed and employed as a training/testing environment to demonstrate the effectiveness of our proposed framework. Our approach may open a new avenue for future research that connects state-of-the-art GNN and MADRL techniques to fleet management problems.

Appendix I
Discussion on Stochastic Policy Update

In this section, we discuss the convergence and optimality issue of our method. Due to the complexity of the fleet management problem, it is difficult to conduct a mathematically rigorous convergence analysis. We can only expect that if an equilibrium state exists for the policy, there will be negative feedback which makes our policy do not deviate from it. For example, if more drivers are sent to a certain road than the equilibrium state policy, the expected reward will be reduced, and the Q value will also be updated to a smaller value. Because the policy is proportional to the increasing function of Q, we can expect that fewer drivers will be sent next time.

On the optimality issue, we can find a simple counter-example. Suppose that two roads (l_1, l_2) are connected in the same direction and the number of drivers and orders are given by Table. III. We assume that all agents are controllable and only two actions are admissible: staying at road 1 (i.e. $l_1 \to l_1$) and moving to road 2 (i.e. $l_1 \to l_2$).

TABLE III: Number of drivers and orders.

	Road 1	Road 2
Number of drivers (N_j)	10	0
Number of orders (N_j^{call})	3	7

For simplicity, suppose that we only have a single step and drivers can be distributed in non-integer value. Then, the reward per unit driver can be expressed as follows:

$$\mathbb{E}[R_j] = \min\left\{1, \frac{N_j^{\text{call}}}{\pi(l_1 \to l_j) N_1}\right\}, \quad j = 1, 2. \tag{15}$$

It is clear that an optimal policy is given by $[\pi(l_1 \to l_1), \pi(l_1 \to l_2)] = [0.3, 0.7]$ and corresponding Q function is given by $[Q(l_1 \to l_1), Q(l_1 \to l_2)] = [1, 1]$. The optimal total reward for all agents will then be 10.

Now consider the stochastic policy update using (8). Assume that initial Q values are set to $[1, 1]$ and α is set to 1. Note that if $N_j = 0$, we do not update its Q value. Fig. 5 shows the converged total reward depending on β. The resulting policy converges to the uniform distribution $[0.5, 0.5]$ as $\beta \to 0$ (for both **Exp** and **Pow** cases) which

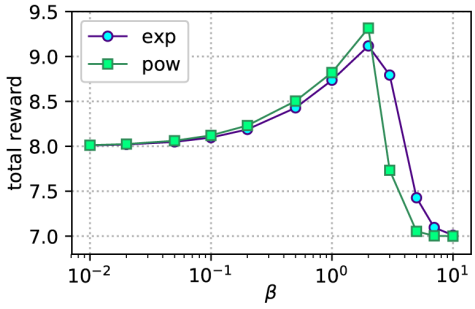


Fig. 5: Converged total reward depending on β with **Exp** and **Pow** Q function.

results total reward to be 8. If $\beta \to \infty$, the stochastic policy converges to the greedy policy $[0, 1]$ and total reward become 7. We can observe that the total reward increases from 8 to nearly 10 (optimal value) and then decreases to 7 as β increases. The optimal β is approximately 2.0. The reason why the reward increases for $\beta \in (0, 2)$ is that our stochastic policy gets more representative power to send more drivers to l_2. The reward decreases for $\beta \in (2, \infty)$ since our policy loses representative power again by becoming excessively greedy.

This example shows that our stochastic policy update does not converge to an optimal policy in general. But it should be noted here that it still gives better results than the standard greedy policy update rule, which is consistent with our intuition.

ACKNOWLEDGMENT

The data were provided by *Kakao Mobility*. We also greatly appreciate Dr. Young Min Kim at Seoul National University for revising the manuscript.

REFERENCES

[1] G. A. Godfrey and W. B. Powell, "An adaptive dynamic programming algorithm for dynamic fleet management, i: Single period travel times," *Transportation Science*, vol. 36, no. 1, pp. 21–39, 2002.

[2] C. Wei, Y. Wang, X. Yan, and C. Shao, "Look-ahead insertion policy for a shared-taxi system based on reinforcement learning," *IEEE Access*, vol. 6, pp. 5716–5726, 2017.

[3] K. Lin, R. Zhao, Z. Xu, and J. 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*. ACM, 2018, pp. 1774–1783.

[4] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning," in *The World Wide Web Conference*, 2019, pp. 983–994.

[5] M. Zhou, J. Jin, W. Zhang, Z. Qin, Y. Jiao, C. Wang, G. Wu, Y. Yu, and J. 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*, 2019, pp. 2645–2653.

[6] J. Ke, F. Xiao, H. Yang, and J. Ye, "Optimizing online matching for ride-sourcing services with multi-agent deep reinforcement learning," *arXiv preprint arXiv:1902.06228*, 2019.

[7] D.-H. Lee, H. Wang, R. L. Cheu, and S. H. Teo, "Taxi dispatch system based on current demands and real-time traffic conditions," *Transportation Research Record*, vol. 1882, no. 1, pp. 193–200, 2004.

[8] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. 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*, 2017, pp. 2151–2159.

[9] K. T. Seow, N. H. Dang, and D.-H. Lee, "A collaborative multiagent taxi-dispatch system," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 607–616, 2009.

[10] A. Alshamsi, S. Abdallah, and I. Rahwan, "Multiagent self-organization for a taxi dispatch system," in *8th international conference on autonomous agents and multiagent systems*, 2009, pp. 21–28.

[11] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. 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*. ACM, 2018, pp. 905–913.

[12] Z. Wang, Z. Qin, X. Tang, J. Ye, and H. Zhu, "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.

[13] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.

[14] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[15] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 554–15 566.

[16] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[17] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.

[19] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial collective intelligence," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[20] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," *arXiv preprint arXiv:1802.05438*, 2018.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[22] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1352–1361.

[23] X. Wang, L. Ke, Z. Qiao, and X. Chai, "Large-scale traffic signal control using a novel multi-agent reinforcement learning," *arXiv preprint arXiv:1908.03761*, 2019.

[24] X. Li, J. Zhang, J. Bian, Y. Tong, and T.-Y. Liu, "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.

[25] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.