Multi-Agent Reinforcement Learning for Urban Crowd Sensing with For-Hire Vehicles

Rong Ding*, Zhaoxing Yang*, Yifei Wei, Haiming Jin†, Xinbing Wang
Shanghai Jiao Tong University, Shanghai, China
Email: {dingrong, yiannis, weiyifei, jinhaiming, xwang8}@sjtu.edu.cn

***Abstract*—Recently, vehicular crowd sensing (VCS) that leverages sensor-equipped urban vehicles to collect city-scale sensory data has emerged as a promising paradigm for urban sensing. Nowadays, a wide spectrum of VCS tasks are carried out by for-hire vehicles (FHVs) due to various hardware and software constraints that are difficult for private vehicles to satisfy. However, such FHV-enabled VCS systems face a fundamental yet unsolved problem of striking a balance between the order-serving and sensing outcomes. To address this problem, we propose a novel graph convolutional cooperative multi-agent reinforcement learning (GCC-MARL) framework, which helps FHVs make distributed routing decisions that cooperatively optimize the system-wide global objective. Specifically, GCC-MARL meticulously assigns credits to agents in the training process to effectively stimulate cooperation, represents agents' actions by a carefully chosen statistics to cope with the variable agent scales, and integrates graph convolution to capture useful spatial features from complex large-scale urban road networks. We conduct extensive experiments with a real-world dataset collected in Shenzhen, China, containing around 1 million trajectories and 50 thousand orders of 553 taxis per-day from June 1st to 30th, 2017. Our experiment results show that GCC-MARL outperforms state-of-the-art baseline methods in order-serving revenue, as well as sensing coverage and quality.**

I. Introduction

Timely, accurate, and comprehensive sensing of urban metrics (e.g., air quality, traffic congestion, infrastructure strain) plays an important role for monitoring a city's health. Traditionally, urban sensing has been carried out with stationary sensors, including loop detectors [1], surveillance cameras [2], and many others. However, these stationary sensors are oftentimes difficult to be scaled up to cover the whole city, because of their prohibitive deployment costs, inconvenient maintenance processes, and limited sensing ranges. To address this issue, *vehicular crowd sensing (VCS)* that uses sensors dedicatedly installed on crowdsourced vehicles or inherently integrated on-board drivers' smartphones arises as a promising sensing paradigm, because it enables the sensors to cover the broad urban areas visited by their hosts.

Nowadays, *for-hire vehicles (FHVs)*, such as taxis and those operated by ride-hailing platforms, are actually the most desirable forces for many VCS systems. One reason is the requirement of specialized hardwares for a wide spectrum of sensing tasks. For instance, air pollution sensing [3] inevitably requires participating vehicles to carry specialized air quality sensors, and networked dashcams [4] with the functionality of streaming recorded videos to servers have to be equipped before collecting front-view driving videos that help monitor real-time traffic conditions. Clearly, a fleet of FHVs managed by a centralized ride-hailing platform or taxi company is much more convenient for the deployment and maintenance of these sensing devices than private vehicles. Even without specialized hardware requirements, software restrictions could also create a barrier that prohibits vehicles other than FHVs from carrying out certain VCS tasks. For example, a ride-hailing platform's navigation service can only rely on its own FHVs for reporting traffic condition updates, as it is exclusively used by those vehicles rather than private ones.

Though promising, today's *FHV-enabled VCS (FVCS)* systems still face a fundamental unsolved problem of how to balance the conflicting objectives of order-serving and sensing. In practice, FHVs tend to concentrate in commercial, business, and tourist areas, where they usually encounter significantly more passenger orders than socio-economically disadvantaged neighborhoods. However, a majority of the VCS tasks inherently requires vehicles to distribute evenly both spatially and temporally, so that enough sensory data could be collected continuously from every corner of the city. Therefore, in this paper, we aim to address the above imperative problem of achieving *satisfactory order-serving revenue*, as well as *sensing coverage and quality* in FVCS systems via a mechanism that helps FHVs make *appropriate route selection decisions*. Making such routing decisions for FHVs is naturally a sequential decision-making task, which aims to maximize the system-wide long-term cumulative reward that integrates both order-serving and sensing outcomes. In what follows, we elaborate on the challenges of design such mechanism, as well as our approaches that address them.

The first challenge comes from the *huge scale* of a real-world FHV fleet, which could usually contain hundreds of vehicles even in a single city. Under such scenario, centralized sequential decision-making mechanisms (e.g, single-agent reinforcement learning) will become infeasible due to the exponential explosion in the state and action spaces. To address this challenge, we leverage the framework of *multi-agent reinforcement learning (MARL)*, which treats each FHV as an agent, and trains a *distributed* decision module for each agent that is able to generate his own routing decisions without any coordination from the system manager.

*Equal contribution.
†Corresponding author.

This work was supported in part by National Key R&D Program of China 2018AAA0101200, in part by NSFC China (No. 61902244, U20A20181, 61960206002, 61822206, 62020106005, 61829201, 62041205, 61532012), in part by Shanghai Municipal Science and Technology Commission Grant 19YF1424600.

However, the above distributed decision-making framework further poses the challenging task of *aligning each agent's local decision with the global objective* of the whole FVCS system. We thus devise a *credit assignment* mechanism, which carefully tailors the reward signals that drive agents towards autonomously learning the routing decisions that cooperatively maximize the global reward. Specifically, we utilize the *difference reward* technique to filter out the noises brought by other agents from the long-term global reward, which is approximated by a central critic in the training process.

In practice, however, the number of FHVs that make routing decisions usually varies over time. Such *variable agent scales* inevitably cause the undesirable variation of the critic's input dimension. We resolve such challenge by meticulously exploring the semantics of the action space, and using the *statistics of actions* which is invariable to the agent scale as the input of the critic, instead of agents' raw actions.

Meanwhile the large-scale road network of a modern city typically has a rather *complicated topology*, making it difficult to extract useful spatial features that are necessary to help the FHVs make appropriate routing decisions. To tackle this challenge, inspired by the recent success of *graph covolutional networks (GCNs)* on capturing graph-structured correlations, we propose to empower our MARL framework with the ability to effectively extract spatial features from complex real-world road networks by carefully integrating it with GCNs.

Overall, we fuse the above components into an integrated *graph convolutional cooperative MARL (GCC-MARL)* framework, which jointly addresses the aforementioned challenges.

In summary, this paper makes the following contributions.

- As far as we know, this work is the first one that jointly optimizes order-serving and sensing outcomes in FVCS systems through a distributed route selection mechanism.
- Technically, we design a novel graph convolutional cooperative MARL framework, which integrates (i) our carefully designed *credit assignment* mechanism that effectively shapes each agent's reward so as to facilitate cooperation, (ii) our *statistic-based* action representation that copes with the variable agent scales, and (iii) the *graph convolution* operation that captures useful spatial features from large-scale urban road networks.
- We conduct extensive experiments with real-world dataset containing 1 million trajectories and 50 thousand orders of 553 taxis per-day from June 1st to June 30th, 2017. The experimental results show that our approach outperforms state-of-the-art baseline methods.

II. Preliminaries

A. System Overview

We consider an FVCS system in a city, where a cloud-based platform manages a set $\mathcal{N} = \{1, 2, \cdots, N\}$ of FHVs for both order-serving and sensing. That is, during their daily practices of serving orders, the FHVs also utilize either pre-installed dedicated sensors or simply those on-board drivers' smartphones to carry out urban sensing tasks. Naturally, an FHV in FVCS system is either *on-service* if it is currently serving orders, or *idle*, otherwise. We discretize the entire time horizon into T equal-length time slots, denoted as $\mathcal{T} = \{1, 2, \cdots, T\}$, and construct the *road network* described in Definition 1 as the spatial representation of a city.

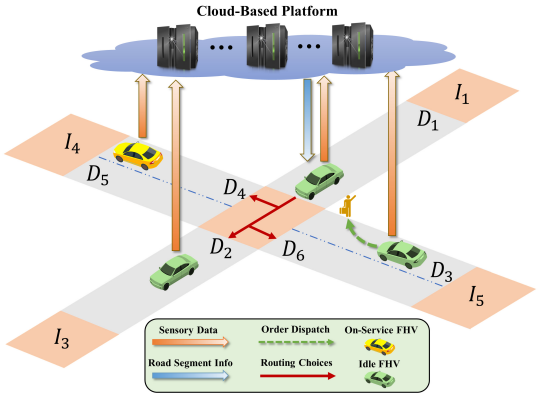


Figure 1: Interactions between the platform and FHVs, where the central intersection is denoted as I_2.

Definition 1 (Road Network)**.** *We define a city's road network as a tuple $(\mathcal{I}, \mathcal{D})$, where $\mathcal{I}$ is the set of all road intersections and $\mathcal{D}$ is the set of all road segments*[1] *that the city has.*

Fig. 1 demonstrates an example of a road network with $\mathcal{I} = \{I_1, I_2, \cdots, I_5\}$ and $\mathcal{D} = \{D_1, D_2, \cdots, D_6\}$, and illustrates the interactions between the platform and FHVs, which we explain in detail as follows.

- At the beginning of each time slot, the platform allocates the passenger orders that have not yet been served to idle FHVs by calling the order dispatch algorithm [5–7]. After picking up passengers, idle FHVs will become on-service ones and start heading to the passengers' destinations.
- When an idle FHV reaches an intersection, the platform sends the information of the surrounding road segments (e.g., distribution of orders and FHVs) to it. Such information is then used by the FHV's local decision module to decide which way to go in the next time slot.
- The FHVs carry out sensing tasks continuously, and will upload the collected sensory data to the platform at the end of each time slot.

In this paper, we focus on designing and training local decision module for each FHV, which helps it make routing decisions. Therefore, tasks such as order dispatch are not the main focus of this paper. In fact, our model is compatible with any existing algorithms to handle these tasks.

B. Problem Description

In this paper, we take the perspective of the platform and aim to achieve both satisfactory order-serving and sensing outcomes. As commonly used in practice, the platform uses the *gross merchandise volume (GMV)* defined in Definition 2 to evaluate FHVs' performance of serving passenger orders.

[1] In our model, a road segment is the piece of a road between two adjacent intersections, and it is directional that the opposite directions of a two-way road are treated as different elements in $\mathcal{D}$.

Definition 2 (GMV). *Let $\mathcal{M}_t$ be the set of orders served by the FHVs in time slot t. The order-serving revenue of the platform in time slot t is $r_t^o = \sum_{e_i \in M_t} p(e_i)$, where $p(e_i)$ denotes the price of each order $e_i \in \mathcal{M}_t$. The overall GMV is defined as the sum of r_t^o along the whole time horizon, i.e.,*

$$\text{GMV} = \sum_{t \in \mathcal{T}} r_t^o = \sum_{t \in \mathcal{T}} \sum_{i: e_i \in M_t} p(e_i). \tag{1}$$

Next, we define the metric *utility of sensing (UoS)* which measures the system-wide sensing performance.

Definition 3 (UoS). *Let n_{jt} be the number of FHVs that traverse the road segment $D_j \in \mathcal{D}$ during time slot t and l_j be the length of D_j. The sensing utility of the platform in time slot t is defined as $r_t^s = \sum_{j: D_j \in \mathcal{D}} l_j \ln(n_{jt} + 1)$. The overall UoS is defined as the sum of r_t^s along the whole time horizon, i.e.,*

$$\text{UoS} = \sum_{t \in \mathcal{T}} r_t^s = \sum_{t \in \mathcal{T}} \sum_{j: D_j \in \mathcal{D}} l_j \ln(n_{jt} + 1). \tag{2}$$

As indicated by Definition 3, UoS jointly captures the *sensing quality of road segments* and *spatio-temporal sensing coverage* to comprehensively evaluate the sensing performance of FHVs. Intuitively, the sensing quality of a piece of road segment is monotonically increasing with the number of FHVs going through it, since a larger number of FHVs will collect more sufficient sensory data. However, when there are already adequate sensory data, further increase of the data volume will bring only minor gain to the sensing quality. The $\ln(n_{jt} + 1)$ term in Equation (2) helps capture the above properties and represent the sensing quality for one unit length of road segment, as the $\ln(\cdot)$ function is non-decreasing and concave. Furthermore, by multiplying $\ln(n_{jt} + 1)$ with the length l_j of road segment D_j and summing it over all the road segments and time slots, we ensure that the UoS defined by Equation (2) covers the sensing quality of every inch of a city's road segments over the entire time horizon.

Next, we integrate the GMV and UoS into the *system utility* as given in Definition 4, which unifies both the order-serving and sensing performances of the entire FVCS system.

Definition 4 (System Utility). *The system utility J is defined as the weighted sum of the GMV and UoS, i.e.,*

$$J = \text{GMV} + \lambda \text{UoS} = \sum_{t \in \mathcal{T}} (r_t^o + \lambda r_t^s), \tag{3}$$

where $\lambda \geq 0$ denotes the importance ratio for UoS and can be set by the platform flexibly.

To obtain satisfactory system utility, FHVs should take reasonable routing choices to appear in a timely manner at the road segments where order-serving and sensing demands exist. Specifically, we choose to optimize the routing decisions for idle FHVs rather than on-service ones for the following two reasons. On one hand, on-service FHVs are supposed to deliver the passengers to their destinations as soon as possible and any explicit deviation from the fastest route will jeopardize passengers' experience. In contrast, the routes of idle FHVs are much more flexible and have a greater potential to be further optimized. On the other hand, a city-scale FHV fleet could usually contain a huge number[2] of idle FHVs at any time instance, whose routing decisions will influence the system utility significantly. Therefore, we aim to design models and algorithms which could help idle FHVs in an FVCS system make appropriate routing decisions that *maximize the overall system utility*.

III. FVCS-POMDP Formulation

Real-world FVCS systems usually operate in highly stochastic environments, where future arrivals and cancellations of passenger orders follow *a priori* unknown patterns. Consequently, it is infeasible to perform a one-shot calculation of the optimal routing choices for all the FHVs and time slots. Under such environments, the platform will have to follow a *sequential decision-making process* that optimizes FHVs' routing choices at each time slot to maximize the expected long-term system utility from the current time slot onwards.

An intuitive approach to address such sequential decision-making problem is to formulate it as a single-agent centralized decision framework, in which the platform is the only agent that decides all the FHVs' routes. However, such framework is infeasible in practice due to the explosion of the state and action spaces caused by the large scale of FHVs. Consequently, we adopt a decentralized decision paradigm that implements an agent at the side of each FHV, who works cooperatively with the other agents to maximize the system utility. We then formulate such multi-agent decision problem as a decentralized partially observable Markov decision process [8] (referred to as *FVCS-POMDP*) as presented in Definition 5.

Definition 5 (FVCS-POMDP). *The FVCS-POMDP is a decentralized partially observable Markov decision process defined by following components.*

- ***Agent:*** *An agent is a local decision module implemented by the platform at an FHV that makes its routing decisions when it is idle. We use $\mathcal{N}$ to represent the agent set.*
- ***State:*** *A state s_t consists of the current time slot t, and the features of each road segment. We construct the feature vector of each road segment $D_i \in \mathcal{D}$ as $\boldsymbol{f}_{it} = (n_{1t}, n_{2t}, n_{3t}, l_i)$, where l_i denotes D_i's length, and n_{1t}, n_{2t}, and n_{3t} denotes respectively the number of orders waiting to be served, the number of on-service FHVs, and the number of idle FHVs on the road segment i in time slot t.*
- ***Observation:*** *At every time slot t, each agent $i \in \mathcal{N}$ receives a local observation o_t^i, which contains the features of its current road segment and those that can be reached within m steps of routing actions[3] by agent i.*
- ***Action:*** *At every time slot t, each agent $i \in \mathcal{N}$ that is leaving its current road segment takes an action a_t^i that indicates the road segment it will enter at the next time slot. We denote*

[2] As is shown in our experiments, more than 40% FHVs are idle on average at each time slot.

[3] We will discuss the details of m in Sec. IV-C.

the set of such agents as $\mathcal{N}_t$, agents' joint actions as $\boldsymbol{a}_t = (a_t^i)_{i\in\mathcal{N}_t}$, and agent i's action space as $\mathcal{A}_t^i$ at time slot t.

- ***Policy:*** *Given observation o_t^i, agent i's policy π^i specifies a probability $\pi^i(a_t^i|o_t^i)$, with which it takes each action $a_t^i \in \mathcal{A}^i$. We denote the joint policy of the agents as $\boldsymbol{\pi} = (\pi^1, \pi^2, \cdots, \pi^N)$.*
- ***State Transition Probabilities****. Given state s_t and agents' joint actions $\boldsymbol{a}_t$, the current state s_t transits to state s_{t+1} according to the probability $P(s_{t+1}|s_t, \boldsymbol{a}_t)$.*
- ***Reward****. At every time slot t, the platform receives an immediate reward $R_t = r_t^o + \lambda r_t^s$, which consists of two parts of team rewards that are decided by agents' joint actions and the current state. The platform aims to maximize the expected long-term cumulative reward $J(\boldsymbol{\pi}) = \mathbb{E}[\sum_{t\in\mathcal{T}} R_t]$.*

Under FVCS-POMDP model, each agent uses its own policy to generate actions in a fully decentralized manner. Furthermore, the team rewards r_t^o and r_t^s are non-decomposable among agents, because they are generated by agents' joint actions, and are affected by the cooperation among agents.

As the state transition probabilities of the FVCS-POMDP are usually *a priori* unknown, we propose to train agents' policies via our novel multi-agent reinforcement learning framework elaborated in Sec. IV.

IV. Proposed GCC-MARL Framework

A. Framework Overview

Our *graph convolutional cooperative multi-agent reinforcement learning (GCC-MARL)* is an actor-critic-based multi-agent reinforcement learning framework with centralized training and decentralized execution [9]. On one hand, a central critic is shared by all agents in the training phase. Specifically, the critic can access the global state and the agents' joint actions, which enables it to capture the mutual influences among agents in a global view and evaluate their cooperation in the system level. On the other hand, each agent possesses a local actor to generate routing actions. As the central critic is not used in the execution phase, agents take actions in a fully decentralized manner in our GCC-MARL framework.

The design details of the actor and critic module in GCC-MARL are given in the following Sections IV-C and IV-D.

B. Reachability Graph

An FHV's current route selection inevitably affects the set of road segments it can reach in the future, and influences the long-term system utility. It is thus critical for the FHVs to know the reachability relationships among road segments. However, a city's road network $(\mathcal{I}, \mathcal{D})$ itself does not explicitly represent the reachability among roads. As a result, we construct a *reachability graph* $\mathcal{C}_t = (\mathcal{V}, \mathcal{E}, \mathcal{F}_t)$ at each time slot t, which effectively captures such reachability information.

Reachability Graph Construction:

- *Node Set ($\mathcal{V}$)*: For each road segment $D_i \in \mathcal{D}$, we construct a corresponding node $v_i \in \mathcal{V}$.
- *Edge Set ($\mathcal{E}$)*: If FHVs leaving road segment D_i are able to enter road segment D_j directly, we construct a directed edge $e_{ij} \in \mathcal{E}$ from node v_i to v_j.

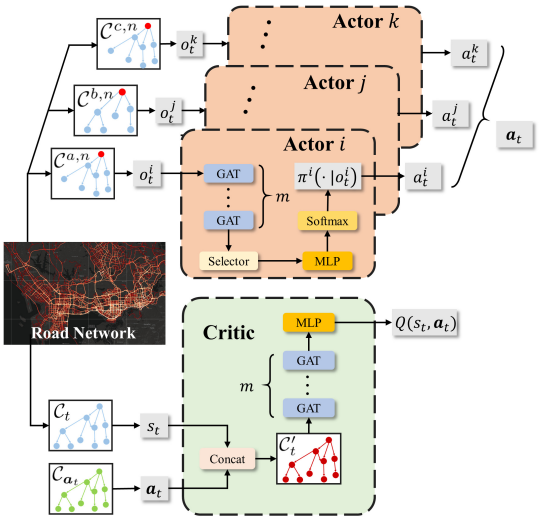
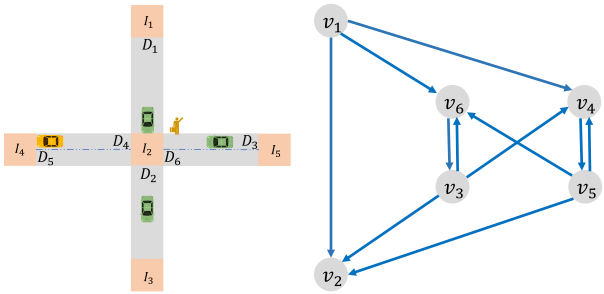


Figure 2: The GCC-MARL framework.

- *Node Features ($\mathcal{F}_t$)*: We set the feature vector of each node $v_i \in \mathcal{V}$ as D_i's feature vector $\boldsymbol{f}_{it}$.

In Fig 3, we show an example of converting a road network into a reachability graph. Clearly, the reachability graph constructed via the above procedure not only captures the reachability relationships among road segments by its edges, but also concentrates meaningful features that characterize the road segments on its nodes. Such property naturally makes the reachability graph conform with the input format and working principles of *Graph Convolutional Networks (GCNs)*, which are expert in extract expressive latent features in graphs. Therefore, in the following Sec. IV-C and IV-D, we use GCNs as the basic building blocks of the actor and critic module.



(a) Road network $(\mathcal{I}, \mathcal{D})$. (b) Reachability graph $\mathcal{C}_t$.

Figure 3: An example of transforming road network into reachability graph.

C. Actor Design

In our GCC-MARL framework, each agent possesses a local actor that generates the routing action. As shown in Fig. 2, at each time slot t, the actor module of an agent i that is about to take an action uses the observation o_t^i as the input to its policy network $\pi^i(\cdot|\cdot)$, and generates a probability distribution $\pi^i(\cdot|o_t^i)$ over its action space, from which the actor module samples the routing action a_t^i.

As aforementioned, the reachability among road segments is essential for an agent to make far-sighted routing actions.

However, feeding the entire reachability graph to the actor module will bring excessive computational overheads. In fact, as will be shown our experiments, letting the actor know the set of road segments that it will reach in a few time slots later is enough for making satisfactory routing decisions. At each time slot t, we use the *n-hop subgraph* of the reachability graph $\mathcal{C}_t$ centered at the road segment D_a where agent i locates, denoted as $\mathcal{C}_t^{a,n} = (\mathcal{V}^{a,n}, \mathcal{E}^{a,n}, \mathcal{F}_t^{a,n})$, as agent i's local observation o_i^t.

n-Hop Subgraph Construction:

- *Node Set* ($\mathcal{V}^{a,n}$): We firstly add the node $v_a \in \mathcal{C}_t$ that corresponds to road segment D_a to $\mathcal{V}^{a,n}$. Next, we perform a Breadth First Search (BFS) starting from v_a along the directed edges of $\mathcal{C}_t$ for n times, and add the nodes that are met in the BFS process into $\mathcal{V}^{a,n}$.
- *Edge Set* ($\mathcal{E}^{a,n}$): We construct the edge set $\mathcal{E}^{a,n}$ as the set of edges in $\mathcal{C}_t$ that are met in the above BFS process.
- *Node Features* ($\mathcal{F}_t^{a,n}$): We set the feature vector of each node $v_i \in \mathcal{V}_t^{a,n}$ as D_i's feature vector $\boldsymbol{f}_{it}$.

The n-hop subgraph $\mathcal{C}_t^{a,n}$ constructed above records the features of all the road segments that agent i may enter in n time slots in the future. Thus, aggregating the features of the nodes in $\mathcal{C}_t^{a,n}$ can help comprehensively evaluate the agent's possible routing choices at the current road segment D_a.

To implement the aggregation process, we stack $m = n-1$ layers of *graph attention networks (GATs)* [10], which are a variant of GCNs that can process directed graphs. Specifically, the input graph $\mathcal{G}^1$ of the 1st GAT layer is the n-hop subgraph $\mathcal{C}_t^{a,n}$, and each layer $l \in \{2, 3, \cdots, m\}$ takes the output $\mathcal{G}^l$ of the $(l-1)$th layer as its input and outputs another graph $\mathcal{G}^{l+1}$.

We show an example of the computation process of one GAT layer in Fig. 4. For each layer l, the input $\mathcal{G}^l$ and output $\mathcal{G}^{l+1}$ have the same graphical structure but different values for the node features. We let $\boldsymbol{h}_p^l \in \mathbb{R}^4$ be the feature vector of each node v_p in the graph $\mathcal{G}^l$, and we thus have $\boldsymbol{h}_p^1 = \boldsymbol{f}_{pt}^{\mathrm{T}}$. Then, GAT layer l calculates the *attention coefficients* α_{pq} between node v_p and each of its 1-hop neighbor node v_q in $\mathcal{G}^l$ as

$$\alpha_{pq} = \frac{\exp\left(\text{LeakyReLU}(\boldsymbol{u}^T[\boldsymbol{W}\boldsymbol{h}_p^l \| \boldsymbol{W}\boldsymbol{h}_q^l])\right)}{\sum_{q' \in \mathcal{V}^{p,1}} \exp\left(\text{LeakyReLU}(\boldsymbol{u}^T[\boldsymbol{W}\boldsymbol{h}_p^l \| \boldsymbol{W}\boldsymbol{h}_{q'}^l])\right)}, \tag{4}$$

where $\|$ denotes the concatenation operation of two vectors. Both $\boldsymbol{W} \in \mathbb{R}^{F \times 4}$ and $\boldsymbol{u} \in \mathbb{R}^{2F \times 1}$ are linear transformation matrices. Equation (4) adopts the nonlinearity activation function LeakyReLU$(\cdot)$ commonly used in neural networks.

Using the attention coefficients as weights, GAT layer l calculates the latent features $\boldsymbol{h}_p^{l+1}$ of node v_p in $\mathcal{G}^{l+1}$ by aggregating the features of its 1-hop neighbor nodes as

$$\boldsymbol{h}_p^{l+1} = \text{LeakyReLU}\left(\sum_{q \in \mathcal{V}^{p,1}} \alpha_{pq} \boldsymbol{h}_p^l\right). \tag{5}$$

After the above operations of one GAT layer, the features of each node is broadcast to its 1-hop neighbors. Thus, through m GAT layers, each node of the input graph is able to aggregate the features of all of its m-hop neighbors. Note that the input of the 1st GAT layer is the n-hop subgraph $\mathcal{C}^{a,n}$ with $n = m+1$ and all the candidate routing road segments are the neighbors of D_a. Such parameter choice enables us to aggregate features of the roads that are m-hops away from each candidate road segment in once computation.

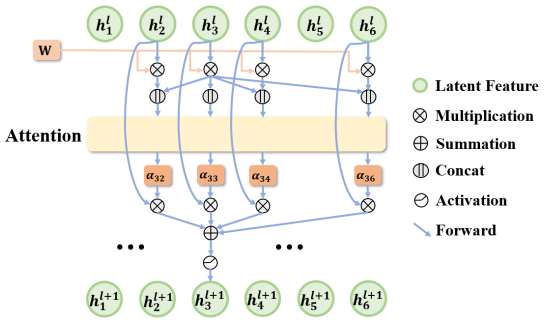


Figure 4: The computation process of the lth GAT layer on node v_3. The input graph is $\mathcal{C}_t^{3,1}$ based on the reachability graph in Fig. 3b.

After we get the output graph of the mth GAT layer, a *selector* is applied to pick the latent features of each candidate routing road segment of agent i's action set $\mathcal{A}_t^i$. These features are then fed into multi-layer perceptrons (MLP), which map them into scalars. Furthermore, a softmax$(\cdot)$ function is applied to generate the stochastic policy that specifies the probability of choosing each candidate routing road segment $D_r \in \mathcal{A}_t^i$ as

$$\pi^i(D_r|o_t^i) = \frac{\exp\left(d(\boldsymbol{h}_r^{m+1})\right)}{\sum_{r':D_{r'} \in \mathcal{A}_t^i} \exp\left(d(\boldsymbol{h}_{r'}^{m+1})\right)}, \tag{6}$$

where $d(\cdot)$ denotes the MLP function. When interacting with the environment, agent i takes action a_t^i according to the probability distribution $\pi^i(\cdot|o_t^i)$. It is worth noting that the actor is able to output the routing action even if the size of its action set varies in different time slots.

D. Critic Design

Our centralized training and decentralized execution framework implements a central critic which accesses the global state s_t and joint actions $\boldsymbol{a}_t$ in the training process, and aims to approximate the state-action value function $Q(s_t, \boldsymbol{a}_t) = \mathbb{E}_{\boldsymbol{\pi}}[\sum_{\tau=t}^{T} R_\tau | s_t, \boldsymbol{a}_t]$ under the current joint policy $\boldsymbol{\pi}$.

A widely applied approach is to use neural networks as the approximator of $Q(s_t, \boldsymbol{a}_t)$. However, simply representing agents' joint actions $\boldsymbol{a}_t$ as a vector that concatenates each agent's action is infeasible in our problem setting. Clearly, the dimension of $\boldsymbol{a}_t$ is huge and could vary significantly across time slots due to the variation of the sets of agents that take actions. Such vector-based representation of $\boldsymbol{a}_t$ is impractical to be processed by neural networks. Hence, it is imperative to represent $\boldsymbol{a}_t$ in a compact but informative data structure. At each time slot t, we construct a novel *action graph* $\mathcal{C}_{\boldsymbol{a}_t} = (\mathcal{V}_{\boldsymbol{a}_t}, \mathcal{E}_{\boldsymbol{a}_t}, \mathcal{F}_{\boldsymbol{a}_t})$ which effectively embeds $\boldsymbol{a}_t$ as follows.

Action Graph Construction:

- *Node Set* ($\mathcal{V}_{\boldsymbol{a}_t}$): We set the node set $\mathcal{V}_{\boldsymbol{a}_t}$ to be the same as the node set $\mathcal{V}$ of the reachability graph $\mathcal{C}_t$.
- *Edge Set* ($\mathcal{E}_{\boldsymbol{a}_t}$): We set the edge set $\mathcal{E}_{\boldsymbol{a}_t}$ to be the same as the edge set $\mathcal{E}$ of the reachability graph $\mathcal{C}_t$.
- *Node Features* ($\mathcal{F}_{\boldsymbol{a}_t}$): The feature f_{it} of each node $v_i \in \mathcal{V}_{\boldsymbol{a}_t}$ captures the statistics of $\boldsymbol{a}_t$, and is set to be the number of

times that the corresponding road segment D_i is picked in the joint actions $\boldsymbol{a}_t$.

The action graph $\mathcal{C}_{\boldsymbol{a}_t}$ has a fixed graphical structure, which is adaptive to the varying and high dimension of $\boldsymbol{a}_t$. Furthermore, $\mathcal{C}_{\boldsymbol{a}_t}$ captures the spatial properties of $\boldsymbol{a}_t$ by inheriting the structure of the reachability graph, which enables the critic to evaluate more accurately the cooperation among agents from the spatial view of the system.

As shown in Fig. 2, at each time slot t, the critic concatenates the features correspond to the same nodes in the reachability graph $\mathcal{C}_t$ and action graph $\mathcal{C}_{\boldsymbol{a}_t}$, and gets a concatenated graph $\mathcal{C}'_t$. The critic takes $\mathcal{C}'_t$ as the input to m stacking GAT layers. Next, a feedforward MLP $g(\cdot)$ maps the outputs of the final GAT layer into scalars. Finally, an average operation aggregates the outputs of the MLP and yields the approximation of the Q function, i.e.,

$$Q(s_t, \boldsymbol{a}_t) \approx \frac{1}{|\mathcal{D}|} \sum_{i:D_i \in \mathcal{D}} g(\boldsymbol{z}_i^{m+1}), \tag{7}$$

where $\boldsymbol{z}_i^{m+1}$ denotes the latent features of road segment D_i output by the final GAT layer.

V. Proposed Training Method

A. Difference Reward-Based Credit Assignment Approach

In the training process, we aim to learn agents' policies that can achieve full cooperation and maximize the system utility. However, the non-decomposability of the team rewards make it difficult for the training algorithm to distinguish the contribution of each individual agent to the system utility, and thus hinders the effective learning of agents' policies. In order to solve such problem, we utilize the *difference reward* technique to perform credit assignment among agents. Such technique shapes the reward signals by filtering out noises brought by other agents from the team reward, and thus helps better assess each agent's contribution. Specifically, we use a simple yet effective difference reward called *Wonderful Life Q-function (WLQ)* [11], which is defined as

$$A_t^i(s_t, \boldsymbol{a}_t) = Q(s_t, \boldsymbol{a}_t) - Q(s_t, \boldsymbol{a}_t^{-i}), \tag{8}$$

where $\boldsymbol{a}_t^{-i}$ refers to the joint actions without agent i. Both $\boldsymbol{a}_t$, and $\boldsymbol{a}_t^{-i}$ are implemented using the action graphs proposed in Sec. IV-D. Clearly, the WLQ $A_t^i(s_t, \boldsymbol{a}_t)$ measures the contribution of agent i to the expected system utility, if it takes action a_t^i. We thus use it as the *advantage function* in the training algorithm described in the following Sec. V-B.

B. Overall Training Algorithm

The overall training algorithm of GCC-MARL is presented in Algorithm 1. At first, the algorithm randomly initializes the parameters of each agent's actor and the critic (line 1). As long as the training has not converged, the algorithm collects and stores the experiences in replay buffers (line 2-12). At each time slot t, each agent i who need to make a routing decision gets observation o_t^i, and takes action a_t^i generated by its actor (line 6). The joint actions $\boldsymbol{a}_t$ for agents are then collected,

ALGORITHM 1: Training Algorithm of GCC-MARL

```
1  Randomly initialize the parameters of each agent i's actor θ_i
     and the central critic φ;
2  while training not finished do
3      Initialize replay buffers U, and B_i for each agent i as ∅;
4      foreach time slot t ∈ {1, 2, ··· , T} do
5          foreach agent i ∈ N_t do
6              Actor takes o_t^i as input and generates action a_t^i;
7          Collect the joint actions a_t and team reward R_t;
8          The critic calculates Q(s_t, a_t);
9          Store tuple (Q(s_t, a_t), R_t) into replay buffer U;
10         foreach agent i ∈ N_t do
11             The critic calculates the advantage function
                 A_t^i(s_t, a_t) by Equation (8) ;
12             Store tuple (π^i(a_t^i|o_t^i), a_t^i, A_t^i(s_t, a_t)) into
                 replay buffer B_i;
13     foreach agent i ∈ ⋃_{t∈T} N_t do
14         Sample K experiences from B_i;
15         Update agent i's actor using Equation (9);
16     Sample all the Q(s_t, a_t) and R_t from U;
17     Update the critic Equation (10);
```

based on which the critic calculates the $Q(s_t, \boldsymbol{a}_t)$ (line 7-8). The algorithm then stores the tuple $\boldsymbol{e}_t = (Q(s_t, \boldsymbol{a}_t), R_t)$ into buffer $\mathcal{U}$ (line 9). Next, the critic calculates the advantage function $A_t^i(s_t, \boldsymbol{a}_t)$ for each agent i that takes action, who then stores the tuple $\boldsymbol{b}_t^i = (\pi^i(a_t^i|o_t^i), a_t^i, A_t^i(s_t, \boldsymbol{a}_t))$ in its buffer $\mathcal{B}_i$ (line 11-12). After that, the algorithm enters the parameter updating processes (line 13-17). For each agent i, we sample K experiences from $\mathcal{B}_i$ and update the parameters of its actor (line 15) using policy gradient $\nabla_{\theta_i} J(\boldsymbol{\pi})$, i.e.,

$$\nabla_{\theta_i} J(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{b}_t^i \sim \mathcal{B}_i}\Big[\nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) A_t^i(s_t, \boldsymbol{a}_t)\Big], \tag{9}$$

where θ_i represents the parameters of agent i's actor. Finally, we sample the experiences from the replay buffer $\mathcal{U}$ and update the parameters of the central critic (line 17) by the TD(1) method to minimize the mean square error loss

$$\mathcal{L}(\phi) = \mathbb{E}_{\boldsymbol{e}_t \sim \mathcal{U}}\Big[\big(Q(s_{t+1}, \boldsymbol{a}_{t+1}) + R_t - Q(s_t, \boldsymbol{a}_t)\big)^2\Big], \tag{10}$$

where ϕ denotes the critic's parameters. Moreover, it is also worth noting that the joint actions at is only used by the critic in the training process. As long as training is completed, only the actor of each agent is employed in the execution process, which operates with only the agent's local observation.

C. Proof of Convergence

Let $\boldsymbol{\theta} = \{\theta_1, \theta_2, \cdots, \theta_N\}$ be the parameters of all agents' policies. Next, we rigorously prove the convergence of Algorithm 1 by firstly proving the following Lemma 1.

Lemma 1. *The WLQ policy gradient of our GCC-MARL algorithm is equivalent to the standard single-agent actor-critic policy gradient which treats $\boldsymbol{a}_t$ as the action of one single agent, i.e., we have*

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\pi}) =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i \in \mathcal{N}_t} \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) A_t^i(s_t, \boldsymbol{a}_t)\Big] \\ =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\boldsymbol{a}_t|s_t) Q(s_t, \boldsymbol{a}_t)\Big]. \end{aligned} \tag{11}$$

Proof. As the the advantage function of our GCC-MARL algorithm is defined as $A_t^i(s_t, \boldsymbol{a}_t) = Q(s_t, \boldsymbol{a}_t) - Q(s_t, \boldsymbol{a}_t^{-i})$, the WLQ policy gradient in our algorithm satisfies

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\pi}) =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i\in\mathcal{N}_t} \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) A_t^i(s_t, \boldsymbol{a}_t)\Big] \\ =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i\in\mathcal{N}_t} \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t)\Big] - \\ & \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i\in\mathcal{N}_t} \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t^{-i})\Big].\end{aligned} \tag{12}$$

Let $d^{\boldsymbol{\pi}}(s_t)$ be the ergodic state distribution defined in [12] and $\boldsymbol{\pi}(\boldsymbol{a}_t^{-i}|s_t)$ be the probability that the joint actions $\boldsymbol{a}_t^{-i}$ is taken by the agents excluding agent i at state s_t. Then, the second term in the right-hand-side of Equation (12) satisfies

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i\in\mathcal{N}_t} \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t^{-i})\Big] \\ =& \sum_{s_t\in\mathcal{S}} d^{\boldsymbol{\pi}}(s_t) \sum_{i\in\mathcal{N}_t} \sum_{\boldsymbol{a}_t^{-i}} \boldsymbol{\pi}(\boldsymbol{a}_t^{-i}|s_t) \\ & \sum_{a_t^i} \pi^i(a_t^i|o_t^i) \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t^{-i}) \\ =& \sum_{s_t\in\mathcal{S}} d^{\boldsymbol{\pi}}(s_t) \sum_{i\in\mathcal{N}_t} \sum_{\boldsymbol{a}_t^{-i}} \boldsymbol{\pi}(\boldsymbol{a}_t^{-i}|s_t) \sum_{a_t^i} \nabla_{\theta_i} \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t^{-i}) \\ =& \sum_{s_t\in\mathcal{S}} d^{\boldsymbol{\pi}}(s_t) \sum_{i\in\mathcal{N}_t} \sum_{\boldsymbol{a}_t^{-i}} \boldsymbol{\pi}(\boldsymbol{a}^{-i}|s_t) Q(s_t, \boldsymbol{a}_t^{-i}) \nabla_{\theta_i} 1 = 0,\end{aligned} \tag{13}$$

where $\mathcal{S}$ denotes the state space. As the policy parameters for each agent are independent, the first term on the right-hand-side of Equation (12) satisfies

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i\in\mathcal{N}_t} \nabla_{\theta_i} \log \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t)\Big] \\ =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\sum_{i\in\mathcal{N}_t} \nabla_{\boldsymbol{\theta}} \log \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t)\Big] \\ =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\nabla_{\boldsymbol{\theta}} \log \prod_{i\in\mathcal{N}_t} \pi^i(a_t^i|o_t^i) Q(s_t, \boldsymbol{a}_t)\Big] \\ =& \mathbb{E}_{\boldsymbol{\pi}}\Big[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\boldsymbol{a}_t|s_t) Q(s_t, \boldsymbol{a}_t)\Big],\end{aligned} \tag{14}$$

where the last equality comes from the fact that $\boldsymbol{\pi}(\boldsymbol{a}_t|s_t) = \prod_{i\in\mathcal{N}_t} \pi^i(a_t^i|o_t^i)$ which holds due to the independence of each agent's policy. By substituting Equations (13) and (14) into Equation (12), we have that $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}}[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\boldsymbol{a}_t|s_t) Q(s_t, \boldsymbol{a}_t)]$, which proves Lemma 1. □

It is worth noting that treating independent actors as a single-agent actor that learns joint actions is key for this proof. Next, we show the following Theorem 1, which states the convergence property of our GCC-MARL algorithm.

Theorem 1. *Let $\nabla_{\boldsymbol{\theta}} J_k(\boldsymbol{\pi})$ be the derivative of $J(\boldsymbol{\pi})$ w.r.t. to $\boldsymbol{\theta}$ at each training iteration k. Then, we have*

$$\lim_{k\to\infty} \inf_k ||\nabla_{\boldsymbol{\theta}} J_k(\boldsymbol{\pi})|| = 0, \text{w.p. } 1. \tag{15}$$

That is, after enough training iterations, the GCC-MARL training algorithm converges to a local maximum of $J(\boldsymbol{\pi})$.

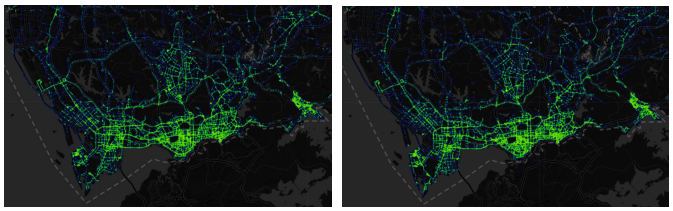
Proof. According to Lemma 1, we have that the WLQ policy gradient of our GCC-MARL algorithm is equivalent to the standard single-agent actor-critic policy gradient given in Equation (11). A single-agent actor-critic algorithm following such gradient is proved to converge to a local maximum of the expected return $J(\boldsymbol{\pi})$ in [13]. Thus, the GCC-MARL training algorithm converges to a local maximum of $J(\boldsymbol{\pi})$. □

VI. Performance Evaluation

A. Simulation Settings

1) Dataset: Our experiments are conducted with around 1,000,000 trajectories and 50,000 orders per-day of 553 taxis from June 1st to June 30th, 2017. We plot the trajectories of all the taxis running in Shenzhen within each 5-minute time period on Shenzhen's road map, and show the result of two representative intervals in Fig. 5. We can easily observe that taxis primarily concentrate on the bottom half of the city where more commercial and business areas exist, but appear much less in the top half which mostly consists of suburban areas. Such observation validates our motivation of helping FHVs make appropriate routing decisions so as to achieve both satisfactory order-serving and sensing outcomes.

2) Simulator Design: To train and evaluate of our GCC-MARL algorithm, we build a simulator for the dynamic urban environment based on a large-scale real-world dataset. Our simulator uses 174 roads, 101 intersections in Nanshan District, Shenzhen, China as the road network, and sets the length of each time slot as 1 minute. The 24 hours from 0:00 to 24:00 in one day is one episode. In the simulator, orders emerge by bootstrapping from dataset and get cancelled if unserved for 3 time slots. As order dispatch is not the focus of this paper, our simulator adopts the simple algorithm that allocates the orders to the idle FHVs on the same road segment uniformly at random. Note that our simulator is compatible with any other sophisticated order dispatch algorithms. After picking up passengers, FHVs will head to destinations through the fastest route. To evaluate the robustness of GCC-MARL in different scenarios, we sample orders with different ratios from our dataset and construct three environment settings, denoted as setting 1, 2 and 3, each of which consists 11109, 33327 and 55545 orders correspondingly.



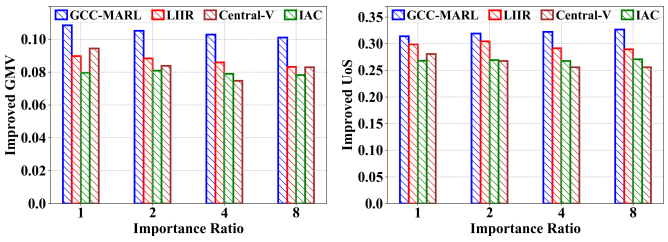
(a) 9:00 to 9:05 (b) 12:00 to 12:05

Figure 5: Taxi trajectories in Shenzhen in two time periods.

B. Baselines and Metrics

We compare GCC-MARL with following strong baselines.

- **Rule-based:** With such method, an FHV always chooses from the candidate road segment that has the maximum average order-serving revenue (i.e., the total order prices divided by the number of idle FHVs).

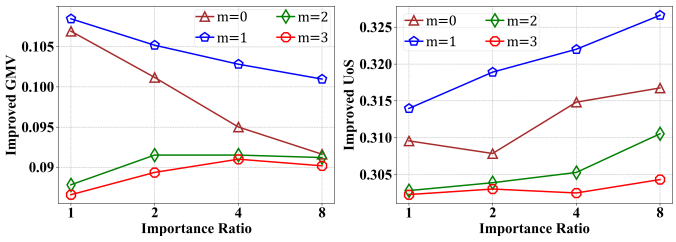


(a) Improved GMV in setting 1. (b) Improved UoS in setting 1.

(c) Improved GMV in setting 2. (d) Improved UoS in setting 2.

(e) Improved GMV in setting 3. (f) Improved UoS in setting 3.

Figure 6: Improved GMV and UoS of different algorithms.



(a) Improved GMV in setting 1. (b) Improved UoS in setting 1.

(c) Improved GMV in setting 2. (d) Improved UoS in setting 2.

(e) Improved GMV in setting 3. (f) Improved UoS in setting 3.

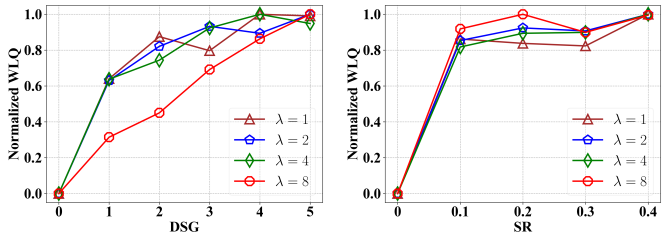
Figure 7: Improved GMV and UoS of GCC-MARL with $m \in \{0, 1, 2, 3\}$.

- **IAC:** IAC is similar to Independent Q-Learning [14] except that it adopts an actor-critic method. An FHV's reward under IAC consists of its order-serving reward (i.e., total prices of the orders it serves), and its sensing reward (i.e., average sensing reward of all FHVs) multiplied by an importance ratio. IAC maintains a decentralized critic for each FHV to evaluate its own long-term reward that it aims to maximize.
- **Central-V:** Central-V [15] learns a centralized critic with a decentralized actor for each agent without filtering out the contributions of other agents in the reward signal.
- **LIIR:** LIIR [16] is a state-of-the-art multi-agent reinforcement learning method for credit assignment, where each agent learns a parameterized intrinsic reward function, and uses the sum of the system utility and intrinsic reward to stimulate each agent to generate cooperative actions.

We define the *improved GMV* of an algorithm as the relative improvement of its performance on GMV compared with the rule-based algorithm, and define the *improved UoS* in the same manner. These two metrics are used to measure the performance of GCC-MARL and the above algorithms.

C. Experiment Results

1) Comparison with Baseline Methods: Fig. 6 shows that our GCC-MARL algorithm outperforms all the baselines in both GMV and UoS in all settings. Specifically, our reasoning of GCC-MARL's superior performance compared with IAC is as follows. Firstly, under IAC all agents aim to maximize their own received rewards, which incurs unnecessary competition among agents that decreases the system utility. Moreover,



(a) Normalized WLQ with different DSG. (b) Normalized WLQ with different SR.

Figure 8: Visualization of Normalized WLQ.

IAC's equal allocation of the overall sensing reward to each agent cannot precisely measure its contribution to the non-divisible team sensing reward.

Compared with GCC-MARL that learns the state-action value function, Central-V only learns the state value function, which makes it unable to discriminatively credit the agents based on their diverse actions. We observe that the performance of Central-V degrades sharply with the increase of the order number. Such observation further implies the inapplicability of Central-V in city-scale FVCS systems.

Although LIIR has been shown to be fairly effective with a small number (e.g., less than 10) of agents [16], it performs worse than GCC-MARL in the setting with hundreds of agents due to the difficulty of learning each agent's intrinsic reward. In contrast, the credit assignment mechanism in GCC-MARL captures the contribution of each agent to the system utility more precisely than the learnt intrinsic reward in LIIR.

2) Impact of the Number of GAT Layers in GCC-MARL: We now evaluate the impact of the number of GAT layers

on the performance of GCC-MARL. Fig. 7 shows the performances of GCC-MARL in different settings with the number of layers $m \in \{0, 1, 2, 3\}$. From these figures, we observe that setting $m = 1$ yields the best performance. The model with zero GAT layer cannot capture any spatial correlation among the road segments and thus performs worse than that with one GAT layer. Interestingly, we observe that $m = 2$ and $m = 3$ yield the worst performances in GMV and UoS among most cases. Reasons behind such observation are as follows. On one hand, the increase of the number of convolutional layers will make GCNs more likely to generate over-smoothed features, which will cause the loss of necessary spatial information and eventually hinder the generation of satisfactory policies. On the other hand, although more GAT layers would propagate features of roads in further expanded neighborhoods, the information from roads that are overly distant may become useless or even noisy for generating policies due to rapidly changing real-world road conditions.

3) Evaluation of Our Credit Assignment Method: As our credit assignment method represented by Equation (8) is the core for training each agent's actor module, we now turn to evaluate its effectiveness. Given the feature vector $\boldsymbol{f}_{it} = (n_{1t}, n_{2t}, n_{3t}, l_i)$ of road segment D_i in time slot t, we define the *demand-supply gap (DSG)* as $\text{DSG} = \max\{n_{1t} - n_{3t}, 0\}$ where $n_{1t} - n_{3t}$ represents the difference between the number of orders and idle FHVs, and define the *sensing ratio (SR)* as $\text{SR} = \frac{l_i}{n_{2t}+n_{3t}}$ where $n_{2t} + n_{3t}$ represents the total number of FHVs on the road segment during the time slot.

Intuitively, a higher credit should be assigned to an FHV, if it drives to a road segment with a larger DSG, since it is more likely to serve orders. Similarly, FHVs driving to a road segment with a larger SR will contribute more to the sensing outcome, and thus should also be assigned a higher credit. In Fig. 8, we show the average normalized WLQ w.r.t. both the DSG and SR with $\lambda \in \{1, 2, 4, 8\}$. The general increasing trend of the curves conforms with the above intuition, and validate the correctness of our credit assignment method.

VII. Related Work

Mobile crowd sensing (MCS) [17], which leverages humans or vehicles equipped with smartphones or dedicated sensors, has become a promising paradigm for large-scale sensing tasks. Recently the research community devoted much effort [3, 18–25] to design and analyze MCS systems. Among them, [18–20, 26, 27] designed incentive mechanisms to stimulate participation, [28, 29] proposed user-interaction networks to enable smooth interaction between users and MCS system, [30–32] focused on preserving users' privacy, and [24, 25] tailored MCS for various application scenarios. Compared with these works that mostly utilized optimization or algorithmic methods, we empower MCS with reinforcement learning (RL) to enable a more autonomous and adaptive decision making mechanism in MCS.

Similar to our work, a recent line of studies [33–37] also adopt RL for decision making in VCS systems. Specifically, [33, 35] used RL to control autonomous vehicles for sensing in different environments, [34, 37] applied RL to jointly manage unmanned vehicles to collect sensing data and charge at multiple charging stations, and [36] leveraged RL to plan paths for robots. However, our work is fundamentally different. On one hand, we consider FHVs that already exist in urban environments as the major forces for VCS, thus avoid the extra costs of purchasing and maintaining dedicated sensing vehicles as in the above works. On the other hand, adopting FHVs for sensing naturally requires to manage hundreds of FHVs to work cooperatively, which is a critical issue that cannot be addressed by the above prior literatures.

Moreover, compared with existing cooperative multi-agent RL (MARL) methods, our GCC-MARL framework itself also bears a fair amount of technical novelty. Specifically, the simplest approach [14] for MARL is to train a policy for each agent that maximizes its own reward, which possibly deviate from optimizing the global objective. Recently the centralized training and decentralized execution framework [9, 38] is widely adopted to evaluate the influence of agents' joint behaviors to the global objective. Based on such framework, [11, 15, 16] stimulated agents to work cooperatively by crediting them with rewards shaped by their contributions to the global reward. These methods are shown to be effective in environments with a small number of agents. However, our problem setting contains orders of magnitudes more and constantly changing number of agents, which will degrade the performance of these methods. Empowered by the novel architecture design and action statistics choice, our GCC-MARL is able to effectively model the joint actions of hundreds of agents even if its dimension varies with time. Furthermore, the integration of the graph attention network [10] enables GCC-MARL to efficiently measure the mutual influence of agents and capture useful spatial features for routing decisions.

VIII. Conclusion

In this paper, we propose a graph convolutional cooperative MARL (GCC-MARL) framework that generates routing decisions for hundreds of FHVs in FVCS systems. Specifically, GCC-MARL is an actor-critic-based algorithm that adopts centralized training and decentralized execution. In GCC-MARL, the decentralized decision module equipped on each FHV avoids the exponential explosion problem in the state and action space that occurs in centralized decision-making mechanisms. With the carefully designed credit assignment approach, the framework stimulates agents to cooperatively maximize the global reward and thus aligns each agent's local decision with the global objective. Furthermore, we design the action statistics to present agents' joint actions, which solves variable agent scales problem for the input of the central critic and efficiently captures the spatial properties of agent's joint actions. Finally, we carefully integrate our MARL framework with graph convolutional networks to effectively extract spatial features from complex real-world networks. Extensive experiments demonstrate that our proposed approach outperforms state-of-art algorithms in both GMV and UoS metrics in different environment settings.

REFERENCES

[1] C. Meng, X. Yi, L. Su, J. Gao, and Y. Zheng, "City-wide traffic volume inference with loop detector data and taxi trajectories," in *SIGSPATIAL*, 2017.

[2] X. Liu, P. Ghosh, O. Ulutan, B. S. Manjunath, K. S. Chan, and R. Govindan, "Caesar: cross-camera complex activity recognition," in *SenSys*, 2019.

[3] K. P. O'Keeffe, A. Anjomshoaa, S. H. Strogatz, P. Santi, and C. Ratti, "Quantifying the sensing power of vehicle fleets," in *Proceedings of the National Academy of Sciences of the United States of America*, 2019, pp. 12 752 – 12 757.

[4] Z. Che, M. G. Li, T. Li, B. Jiang, X. Shi, X. Zhang, Y. Lu, G. Wu, Y. Liu, and J. Ye, "D2-city: A large-scale dashcam video dataset of diverse traffic scenarios," in *ArXiv*, 2019.

[5] J. Munkres, "Algorithms for the assignment and transportation problems," in *Journal of the society for industrial and applied mathematics*, 1957, pp. 32–38.

[6] 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 *KDD*, 2018.

[7] X. Tang, Z. T. Qin, F. Zhang, Z. Wang, Z. Xu, Y. Ma, H. Zhu, and J. Ye, "A deep value-network based approach for multi-driver order dispatching," in *KDD*, 2019.

[8] T. Jaakkola, S. P. Singh, and M. I. Jordan, "Reinforcement learning algorithm for partially observable markov decision problems," in *NIPS*, 1995.

[9] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *NIPS*, 2016.

[10] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[11] D. T. Nguyen, A. Kumar, and H. C. Lau, "Credit assignment for collective multiagent RL with global rewards," in *NIPS*, 2018.

[12] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *NIPS*, 1999.

[13] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *NIPS*, 1999.

[14] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *ICML*, 1993.

[15] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI*, 2018.

[16] Y. Du, L. Han, M. Fang, J. Liu, T. Dai, and D. Tao, "LIIR: learning individual intrinsic reward in multi-agent reinforcement learning," in *NIPS*, 2019.

[17] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," in *IEEE Communications Magazine*, 2011, pp. 32–39.

[18] J. Xu, Z. Rao, L. Xu, D. Yang, and T. Li, "Incentive mechanism for multiple cooperative tasks with compatible users in mobile crowd sensing via online communities," in *IEEE Transactions on Mobile Computing, TMC*, 2020, pp. 1618–1633.

[19] J. Lin, M. Li, D. Yang, and G. Xue, "Sybil-proof online incentive mechanisms for crowdsensing," in *INFOCOM*, 2018.

[20] L. Xiao, Y. Li, G. Han, H. Dai, and H. V. Poor, "A secure mobile crowdsensing game with deep reinforcement learning," in *IEEE Trans. Inf. Forensics Secur.*, 2018, pp. 35–47.

[21] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing," in *Mobicom*, 2012.

[22] S. Yang, K. Han, Z. Zheng, S. Tang, and F. Wu, "Towards personalized task matching in mobile crowdsensing via fine-grained user profiling," in *INFOCOM*, 2018.

[23] M. Xiao, J. Wu, L. Huang, R. Cheng, and Y. Wang, "Online task assignment for crowdsensing in predictable mobile social networks," in *IEEE Transactions on Mobile Computing, TMC*, 2017, pp. 2306–2320.

[24] Y. Gao, W. Dong, K. Guo, X. Liu, Y. Chen, X. Liu, J. Bu, and C. Chen, "Mosaic: A low-cost mobile sensing system for urban air quality monitoring," in *INFOCOM*, 2016, pp. 1–9.

[25] Y. Hu, G. Dai, J. Fan, Y. Wu, and H. Zhang, "Blueaer: A fine-grained urban pm2.5 3d monitoring system using mobile sensing," in *INFOCOM*, 2016.

[26] H. Jin, L. Su, B. Ding, K. Nahrstedt, and N. Borisov, "Enabling privacy-preserving incentives for mobile crowd sensing systems," in *ICDCS*, 2016.

[27] H. Jin, L. Su, and K. Nahrstedt, "Centurion: Incentivizing multi-requester mobile crowd sensing," in *INFOCOM*, 2018.

[28] M. H. Cheung, F. Hou, and J. Huang, "Make a difference: Diversity-driven social mobile crowdsensing," in *INFOCOM*, 2017.

[29] C. Jiang, L. Gao, L. Duan, and J. Huang, "Scalable mobile crowdsensing via peer-to-peer data sharing," in *IEEE Transactions on Mobile Computing, TMC*, 2018, pp. 898–912.

[30] J. Lin, D. Yang, M. Li, J. Xu, and G. Xue, "Frameworks for privacy-preserving mobile crowdsensing incentive mechanisms," in *IEEE Transactions on Mobile Computing, TMC*, 2018, pp. 1851–1864.

[31] H. Wu, L. Wang, G. Xue, J. Tang, and D. Yang, "Enabling data trustworthiness and user privacy in mobile crowdsensing," in *IEEE/ACM Transactions on Networking, TON*, 2019, pp. 2294–2307.

[32] X. Gong and N. B. Shroff, "Truthful mobile crowdsensing for strategic users with private data quality," in *IEEE/ACM Transactions on Networking, TON*, 2019, pp. 1959–1972.

[33] C. H. Liu, Z. Dai, Y. Zhao, J. Crowcroft, D. O. Wu, and K. Leung, "Distributed and energy-efficient mobile crowdsensing with charging stations by deep reinforcement learning," in *IEEE Transactions on Mobile Computing, TMC*, 2019.

[34] C. H. Liu, C. Piao, and J. Tang, "Energy-efficient uav crowdsensing with multiple charging stations by deep learning," in *INFOCOM*, 2020.

[35] C. H. Liu, Z. Dai, H. Yang, and J. Tang, "Multi-task-oriented vehicular crowdsensing: A deep learning approach," in *INFOCOM*, 2020.

[36] Y. Wei and R. Zheng, "Informative path planning for mobile sensing with reinforcement learning," in *INFOCOM*, 2020.

[37] C. H. Liu, Y. Zhao, Z. Dai, Y. Yuan, G. Wang, D. Wu, and K. K. Leung, "Curiosity-driven energy-efficient worker scheduling in vehicular crowdsourcing: A deep reinforcement learning approach," in *ICDE*, 2020.

[38] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning," in *ICML*, 2018.