Deep Multi-Agent Reinforcement Learning for Online Order-Dispatching and Repositioning of Heterogeneous Drivers

Ilgin Dogan* **Yiduo Huang***
University of California, Berkeley
* All authors contributed equally.

Extended Abstract

During the COVID-19 pandemic, the demand for food delivery has surged. Food delivery companies are dealing with millions of orders per hour. They need to assign orders to available drivers efficiently in order to better manage the profits and availability of the drivers. However, these short-term order-dispatching decisions might cause drivers to end up in remote areas (without any nearby orders) after finishing their assigned orders. Thus, the companies need to also solve the problem of repositioning the drivers that will have a crucial long-term impact on the service level for future orders. The order-dispatching and driver-repositioning (a.k.a. fleet management) tasks jointly result in a difficult problem due to dynamic number of active drivers and number of orders in the system.

We study solving the online order-dispatching and multiple driver-repositioning tasks in a food delivery platform in a reliable and efficient way using a deep multi-agent reinforcement learning (MARL) framework with heterogeneous agents. We refer to our problem as the Online Order-Dispatching and Multi-Driver Repositioning Problem (Online-ODMDRP). The online-ODMDRP includes two different but correlated tasks: order-dispatching and driver-repositioning tasks. The repositioning task has two goals: i) to relocate the idle drivers to help them find new customers if they have no near-by orders or newly enter the system, ii) to dispatch the drivers based on their preferences of exiting the system after completing the existing orders.

The major contributions of our work to the Online-ODMDRP can be summarized as follows. (1) We introduce a realistic MARL environment that is motivated by the needs of real-world food-delivery platforms. To achieve this, we consider a setting where the drivers are not homogeneous as opposed to the existing approaches in the literature. While solving the order-dispatching task, we consider the personal constraints and preferences of drivers such as the location of their homes and their vehicle types. In addition, we provide a reward structure that is aimed to capture the effect of the long distances travelled by the drivers to pick-up the orders from the restaurants both on the long-term customer satisfaction level and on the returns obtained by drivers. (2) We build an efficient food-delivery simulator and provide empirical analyses based on three Actor-Critic (AC) methods with different neural network architectures on instances of the Online-ODMDRP. These three methods are Distributed AC, Centralized AC, and Shared-Experience AC (Christianos et al., 2020).

Our online-ODMDRP framework inherently includes very complex and dynamic interactions between multiple agents. Hence, it is important for the agents to be able to implicitly coordinate with each other while interacting with the environment synchronously in order to maximize their returns. However, the distributed and centralized executions of this framework take away any opportunity for the agents to effectively coordinate with each other. As a result, we observe that SEAC outperforms the other approaches. However, we find that the highly dynamic and stochastic nature of our environment and the complex interactions between the drivers result in high fluctuations in the performance of the AC-based methods. We discuss some potential causes for this instability and aim to perform further analyses to stabilize our framework as a future work.

CS 285 Deep Reinforcement Learning – Final Project, UC Berkeley, Fall 2020.

1 Introduction

During the COVID-19 pandemic, the demand for food delivery has surged. Food delivery companies are dealing with millions of orders per hour. They need to assign orders to available drivers efficiently in order to better manage the profits and availability of the drivers. However, these short-term order-dispatching decisions might cause drivers to end up in remote areas (without any nearby orders) after finishing their assigned orders. Thus, the companies need to also solve the problem of repositioning the drivers that will have a crucial long-term impact on the service level for future orders. The order-dispatching and driver-repositioning (a.k.a. fleet management) tasks jointly result in a difficult problem due to dynamic number of active drivers and number of orders in the system.

We study solving the online order-dispatching and multiple driver-repositioning tasks in a food delivery platform in a reliable and efficient way using a deep multi-agent reinforcement learning (MARL) framework with heterogeneous agents. We refer to our problem as the Online Order-Dispatching and Multi-Driver Repositioning Problem (Online-ODMDRP). The online-ODMDRP includes two different but correlated tasks: order-dispatching and driver-repositioning tasks. The repositioning task has two goals: i) to relocate the idle drivers to help them find new customers if they have no near-by orders or newly enter the system, ii) to dispatch the drivers based on their preferences of exiting the system after completing the existing orders. The repositioning task is solved based on the structure of the city plan and the distribution of the existing orders in the buffer.

The major contributions of our work to the Online-ODMDRP can be summarized as follows.

1. We introduce a realistic MARL environment that is motivated by the needs of real-world food-delivery platforms.
   - 1a) We consider the effect of travel distance on the long-term demand. The distances travelled by drivers have significant effects on both drivers and customers. Since drivers are not getting paid during the time of travelling to a pick-up location, we suggest that minimizing the duration of these travels will contribute to the revenues earned by each driver. In addition, minimizing the waiting time of customers will improve the order response (estimated delivery) time and customer experience in the long-term. We incorporate this effect into our reward structure by assuming that the following costs are incurred to a driver: i) a one-step travel cost based on driver's most recent movement, ii) a food on-board cost proportional to travel time of an order (from its pick-up to delivery). We also assume that if an order cannot be delivered within a specified time window, then it is lost from the system as a penalty.
   - 1b) We consider a realistic problem setting with heterogeneous drivers in contrast to the existing approaches that identify all drivers as homogeneous agents. We add additional individual-specific features to characterize each driver: i) information about the vehicle of a driver including its speed and available carrying capacity, ii) location of the house/original location that a driver will return to exit the system.
2. We build an efficient food-delivery simulator and provide empirical analyses based on Actor-Critic (AC) methods with different neural network structures on instances of the Online-ODMDRP.

2 Related Work

The online-ODMDRP constitues a difficult MARL problem due to the non-stationarity caused by concurrently learning agents and the dynamic structure of the state space and population size. Existing studies often provide heuristic solutions to solve this problem. A growing number of studies in the literature develop RL algorithms to solve the order-dispatching task (see for example, Li et al., 2019; Tang et al., 2019; Ke et al., 2019; Zhou et al., 2019; Yang et al., 2020) and driver-repositioning task (see for example, Gao et al., 2018; Lin et al., 2018; Yang et al., 2018; Wen et al., 2017).

As a single-agent RL framework, Wang et al. (2018) use deep Q-learning with knowledge transfer for learning order dispatching strategies form the perspective of a single driver in a ride-sharing platform. Similarly, Xu et al. (2018) formulate the order-dispatching task as a single-agent setting by training a centralized/shared neural network architecture. However, centralized execution of the online order-dispatching and driver-repositioning tasks are ineffective to capture the stochastic demand-supply dynamics and complex interactions between a large number of concurrently learning agents.

Tan (1993) provides an analysis and comparison between independent Q-learning and a cooperative counterpart in different settings, and provided an empirically proof of the learning efficiency of the cooperative Q-learning. More recently, Li et al. (2019) solve the order-dispatching task by proposing "independent" and "cooperative" AC methods in which they use a mean-field approximation for the drivers in the same neighborhood. A distributed deep Q-network (DQN)-based framework (in which each driver is trained independently) is proposed for the driver-repositioning task by Oda and Joe-Wong (2018) and for the order-dispatching task by Al-Abbasi et al. (2019).

While distributed execution of the online order-dispatching and driver-repositioning tasks are more efficient and easy in implementation, they completely ignore the complex interactions and competition between multiple agents resulting in sub-optimal solutions. Therefore, we refer to a recently proposed MARL algorithm, called the Shared-Experience Actor-Critic (SEAC) that is empirically shown to outperform both the distributed and centralized approaches in several multi-agent environments (Christianos et al., 2020). We adopt SEAC into our realistic food-delivery environment and conduct an empirical analyses using our efficient food-delivery simulator.

3 MARL Environment

The online-ODMDRP can be formulated as a Partially-Observable Markov Decision Process (POMDP) characterized by the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}, N)$ whose components are described below in detail. The goal of the online-ODMDRP is to maximize the cumulative reward obtained from the MARL environment at the end of a finite time horizon. We assume that the environment provides reward only for the order-dispatching actions. To clarify, we consider a 2-dimensional city environment with many restaurants and drivers. Each driver is considered as an agent in the MARL framework.

- N: Number of active heterogeneous drivers/agents in the current system. Since activeness of drivers occurs via a random process, we assume that N is a dynamic input.
- S: Environment state space that is not fully observable to the agents. We denote the system state at time t as $s_t \in \mathcal{S}$ and it consists of $s_t = (d_t^1, \ldots, d_t^N, C_t, s_t^{\text{city}})$ where d_t^i is the state (i.e. feature) vector for agent i at time t, C_t is the current buffer of unassigned customer orders at time t, and s_t^{city} is the current state of the city (i.e. traffic density) at time t which is assumed to evolve independent from the current states of the agents.
  - Each customer order k, $c^k \in C_t$, is defined by the tuple $c^k = (x_{pick}^k, x_{drop}^k, \rho^k, pt^k)$ where
    - x_{pick}^k is the pick-up location (i.e. restaurant) of order k
    - x_{drop}^k is the drop-off location (i.e. customer's location) of order k
    - ρ^k is the fee that a driver earns from order k
    - w^k is the latest possible pick-up time window for order k. We assume that if the realized pick-up time of an order is not within w^k, then this order is lost and removed from the system.
  - Each agent i is characterized by the tuple $d_t^i = (x_t^i, v^i, c_t^{i,\text{pend}}, c_t^{i,\text{cand}_1}, \ldots, c_t^{i,\text{cand}_M}, h^i)$ where
    - x_t^i is the current location of agent i at time t that is specified by a 2-dimensional tuple
    - v_t^i is the vehicle information (speed and available capacity) of agent i at time t
    - $c_t^{i,\text{pend}} \in U_t^i \in C_t$ is the pending order (order to be picked) for agent i at time t where U_t^i is the set of orders on board that has been already picked up by agent i at time t
    - $\{c_t^{i,\text{cand}_1}, \ldots c_t^{i,\text{cand}_M}\} \subset C_t$ is the set of candidate orders that we consider as to be the next potential pending order at the next time step
    - $\kappa_t^i := \{[x(u_{t,j}^i), y(u_{t,j}^i)], j = 1, \ldots, |U_t^i|\}$ is the optimal path to fulfill the current orders on board where $[x(u_{t,j}^i), y(u_{t,j}^i)]$ is the drop-off location of order $u_{t,j}^i$. To make sure that the length of input is fixed, we pad this vector with the last drop-off location and time on the current trajectory so that the length of κ_t^i is equal to the maximum vehicle capacity of agent i.
    - h^i is the location of the house/original destination of agent i. We assume that the agent can only accept the orders whose pick-up/drop-off locations are within a certain radius of h^i.
  - s_t^{city} is the feature of the city. We approximate it based on the historical average demand distribution over the city.

- $\mathcal{O} = O^1 \times O^2 \times \ldots \times O^N$: Joint observation space where O^i is the private observation available to agent i. We assume that an agent is not able to observe the state of the other agents. Thus, we define the observation of agent i at time t as $o_t^i = (d_t^i, s_t^{\text{city}}) \in O^i$.
- $\mathcal{A} = A^1 \times A^2 \times \ldots \times A^N$: Joint action space for N agents. An action $a_t^i \in A^i = \{0, 1, \ldots, M\}$ is defined as
  - $a_t^i = k \geq 1$ if agent i decides either to have a new pending order chosen from its candidate orders (order dispatching task) or to dispatch to a new area (repositioning task)
  - $a_t^i = 0$ if agent i keeps following its current trajectory.
- $\mathcal{T}$: State transition dynamics. At each time point, after assigning the orders to agents and updating the paths of drivers, we update the system state components for agent i according to Algorithm 1. The city features are updated according to an independent random process.
- $\mathcal{R} = R^1 \times R^2 \times \ldots \times R^N$: Joint reward space for N agents. We assume that an agent obtains the reward associated with an order at the time step when the order is delivered. In addition, a one-step travel cost is incurred to an agent at each time step. Our reward structure is mainly inspired from Al-Abbasi et al. (2019). We compute the reward associated with order k, $c^k \in U_t^i$, that agent i earns at time t based on the delivery of this order as follows:

$$r_t^i(d_t^i, a_t^i) = \beta_1 \rho^k - \beta_2 |U_t^i| - \beta_3 J^i \tag{1}$$

where J^i is the one-step travelling cost, i.e., unit amount of oil consumed at a single time step by agent i.

4 Actor-Critic Frameworks

4.1 Baselines and Neural Network Architectures

Distributed/Independent Actor-Critic (DAC): This approach assumes independent learning for each agent, i.e. a separete policy network is trained for each agent using only its own experience. DAC implements an independent AC algorithm for each agent by directly optimizing the following policy loss (2) and value loss (3) for agent i:

$$\mathcal{L}(\theta^i) = \log \pi_{\theta^i}(a_t^i | o_t^i)(r_t^i(d_t^i, a_t^i) + \gamma V_{\theta^i}^{\pi}(o_{t+1}^i) - V_{\theta^i}^{\pi}(o_t^i)) \tag{2}$$

$$\mathcal{L}(\phi^i) = \frac{1}{2} \left|\left| V_{\phi^i}^{\pi}(o_t^i) - y_t^i \right|\right| \text{ where } y_t^i = r_t^i(d_t^i, a_t^i) + \gamma V_{\theta^i}^{\pi}(o_{t+1}^i) \tag{3}$$

where the policy learned by agent i is parametrized by θ^i and the value function $V_{\theta^i}^{\pi}(o_t^i)$ of agent i is parametrized by ϕ^i.

In our implementation, we train N actor and N critic networks. The input of the actor network for agent i is (o_t^i, a_t^i) and the output is a scalar. We feed the output of the actor network into a Boltzmann softmax selector

$$\pi(c_t^{i,cand_k}, |o_i^t) = \frac{\exp(\mu_i(o_i^t, c_t^{i,cand_k}))}{\sum_{m=1}^{M} \exp(\mu_k(o_i^t, c_t^{i,cand_m}))}, \ \forall k \in \{1, \ldots, M\} \tag{4}$$

to decide the next pending order for agent i among the candidate orders $c_t^{i,cand_m},\ m \in \{1, \ldots, M\}$.

Centralized/Shared-Network Actor-Critic (CAC): This approach implements a single actor-critic algorithm and trains a single shared policy network using the on-policy data of all agents. The environment takes the concatenation of all drivers' observations as its input, and gives policies to all the drivers.

In our implementation, the input size of the shared network is $N * \dim(d_t^i)$ (note that the dimension of the state tuple is same for each agent $i \in \{1, \ldots, N\}$) and the output size is $N * M$ where M is the number of candidate orders given in the state tuple of an agent. The output of the shared network is the logits for the categorical distribution.

4.2 Shared-Experience Actor-Critic (SEAC)

As mentioned in Section 2, we adopt SEAC (see Christianos et al., 2020) algorithm into our food-delivery MARL environment. SEAC operates similarly to DAC but updates the actor and critic parameters of an agent by combining gradients computed on the agent's experience with weighted gradients computed on other agents' experience. While training an agent, SEAC uses on-policy trajectory generated by that agent and off-policy trajectory generated by the other agents by implementing importance sampling.

More specifically, SEAC optimizes the following policy loss (5) and value loss (6) for agent i:

$$\begin{aligned}\mathcal{L}(\theta^i) &= \log \pi_{\theta^i}(a_t^i|o_t^i)(r_t^i(d_t^i,a_t^i) + \gamma V_{\theta^i}^{\pi}(o_{t+1}^i) - V_{\theta^i}^{\pi}(o_t^i)) \\ &\quad + \lambda \sum_{k\neq i} \frac{\pi_{\theta^i}(a_t^k|o_t^k)}{\pi_{\theta^k}(a_t^k|o_t^k)} \log \pi_{\theta^i}(a_t^k|o_t^k)(r_t^k(d_t^k,a_t^k) + \gamma V_{\theta^i}^{\pi}(o_{t+1}^k) - V_{\theta^i}^{\pi}(o_t^k)) \qquad (5)\end{aligned}$$

$$\begin{aligned}\mathcal{L}(\phi^i) &= \frac{1}{2}\left|\left|V_{\phi^i}^{\pi}(o_t^i) - y_t^{i,i}\right|\right| + \frac{\lambda}{2}\sum_{k\neq i}\frac{\pi_{\theta^i}(a_t^k|o_t^k)}{\pi_{\theta^k}(a_t^k|o_t^k)}\left|\left|V_{\phi^i}^{\pi}(o_t^k) - y_t^{i,k}\right|\right| \\ &\text{where } y_t^{i,k} = r_t^k(d_t^k,a_t^k) + \gamma V_{\theta^i}^{\pi}(o_{t+1}^k) \qquad (6)\end{aligned}$$

where the hyperparameter λ is weighs the effect of experience of other agents. Christianos et al. (2020) reports that the performance of SEAC is robust with respect ot the values of λ. We use $\lambda = 1$ in our experiments as suggested by Christianos et al. (2020).

5 Efficient Simulator Design for Online-ODMDRP

We develop an efficient simulator based on the realistic MARL environment that we introduced in Section 3 for the online order-dispatching and driver-repositioning tasks in a food-delivery system. We provide a pseudocode for our simulator in Algorithm 1. We implement all AC methods described in Section 4 using this simulator.

In Algorithm 1, we first set the maximum number of candidate orders for each driver and initialize the state and number of active agents in the sytem (lines 1 and 2). At each time step $t \in \mathcal{T}$, we first collect the new customer orders (line 3) and then each agent i determines its action a_i^t, updates its estimated trajectory and observe the reward associated with a_i^t (lines 5 - 15).

Algorithm 1 Online-ODMDRP Simulator

1: Set: M, N
2: Initialize: States of the drivers
3: **for** $t \in \mathcal{T}$ **do**
4: Get all customer orders at time t
5: **for** each driver $i \in \{1,\ldots,N\}$ **do**
6: Check if driver i can pick-up its pending order or drop-off any orders on board
7: **if** can pick-up or drop-off orders **then** update U_t^i
8: Assign the closest M orders as the candidate orders for driver i: $c_t^{i,\text{cand}_1},\ldots,c_t^{i,\text{cand}_M}$
9: Get the action a_t^i to decide if driver i will accept a new order ($a_t^i \geq 1$) or not ($a_t^i = 0$)
10: **if** $a_t^i = k \geq 1$ **then**
11: Update the current pending order $c_t^{i,\text{pend}} = c_t^{i,\text{cand}_k}$
12: Update the trajectory of driver i by solving a TSP.
13: Compute the reward associated with accepting the new order c_t^{i,cand_k}
14: **else**
15: Compute the reward associated with the current trajectory
16: Update the environment according to the updated the trajectory

6 Numerical Experiments

In our numerical experiments, we work with synthetic data. We consider a 10×10 grid-based city using the Manhattan distance metric. There are 15 restaurants located mostly in the corner locations of the city. We generate orders for restaurants according to a Poisson process. For each active restaurant, we generate the destination of the orders with a Uniform distribution over the city plane. Given the orders on board (that has been already picked-up), we find the optimal path of a driver by solving a Traveling Salesman Problem (TSP) using a Dynamic Programming approach. We use Python3 and PyTorch to implement our Online-ODMDRP framework.

We perform tuning for the following hyperparameters of our MARL environment:

- The maximum number of candidate orders for an agent (M)
- The maximum available vehicle capacity of each agent
- The radius around the original destination of an agent (so that it can accept only the orders whose pick-up/drop-off locations are within that radius)
- Maximum number of orders kept in the system buffer

In our preliminary experiments, we observe that CAC has the worst performance by providing much lower returns from DAC and SEAC in all scenarios. That is not surprising since the centralized execution of the online-ODMDRP framework takes away any opportunity for the drivers to effectively coordinate with each other and improve their returns. Therefore, we only present the results for DAC and SEAC in this paper.

We test DAC and SEAC in different multi-agent ($N = 3$) environments with different values of the vehicle capacities of agents. We report the learning curves for these experiments in Figures 2 and 3. In Figure 1, we present the performance of DAC for a single-agent environment ($N = 1$) with different vehicle capacity values.We observe similar patterns for different algorithms across experiments. We find that the mean evaluation returns are lower in the experiments with higher vehicle capacities even in single-agent environments (see Figure 1). We explain this by our reward structure in which we penalize the drivers if they cannot deliver the orders on board quickly.

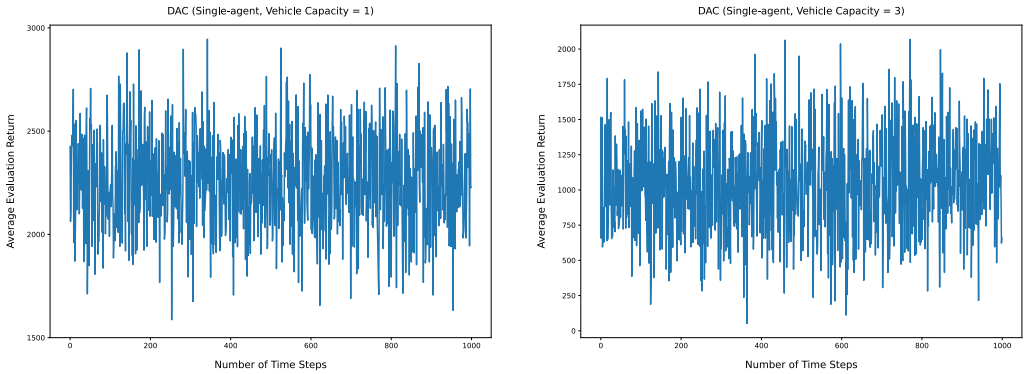

Figure 1: Performance of DAC for single-agent environment with different vehicle capacities.

6.1 Results & Analysis

Although the resulting learning curves are quite fluctuating, SEAC yields the best learning process with higher mean evaluation returns and smaller standard deviations of evaluation returns. Our online-ODMDRP framework inherently includes very complex and dynamic interactions between multiple agents. Hence, it is important for the agents to be able to implicitly coordinate with each other while interacting with the environment synchronously in order to maximize their returns. However, both the centralized and distributed executions of this framework take away any opportunity for the agents to effectively coordinate with each other. Therefore, our results match with our expectations as SEAC considers combining the local gradients to improve the ability of the agents to learn how to coordinate in such a highly stochastic system.

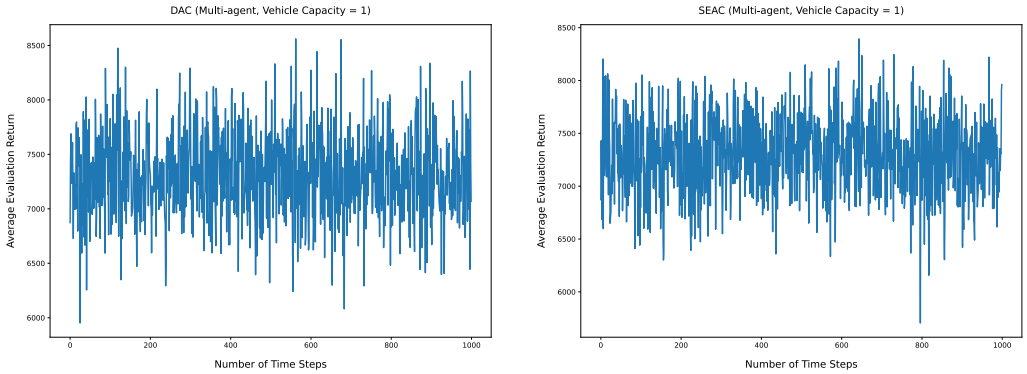

Figure 2: Comparison of DAC and SEAC for multi-agent environment with vehicle capacity = 1.

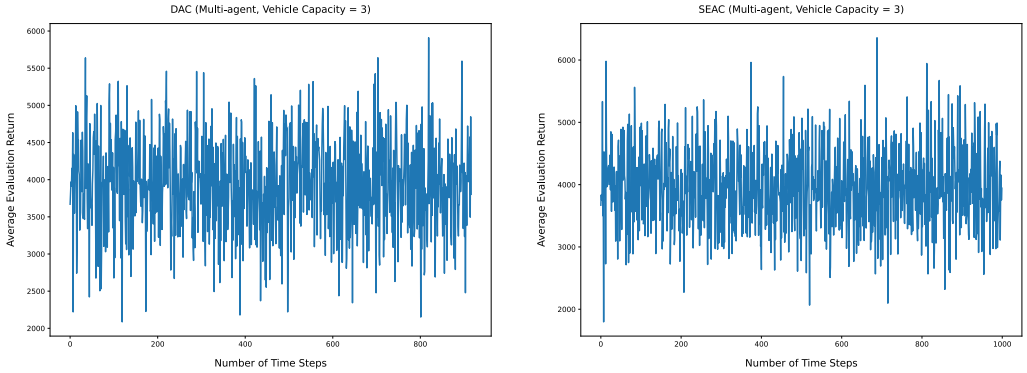

Figure 3: Comparison of DAC and SEAC for multi-agent environment with vehicle capacity = 3.

However, we observe that none of the approaches can actually start learning and improve the returns in 1000 iterations. Hence, we would like to note here some potential causes for our current results. Firstly, we observe that our environment is highly dynamic and the process of order generating is highly stochastic. Secondly, we have a very high-dimensional state space which causes the critic network to take a long time to find a good approximation of the value function for an agent. As a result of a poorly approximated value function, the actor network also generates policies with poor performance. Lastly, we believe that we could not conduct hyperparameter tuning sufficiently due to high computational requirements in terms of environment steps. We believe that the performance of the algorithms might improve and stabilize more after conducting a more adequate hyperparameter tuning.

7 Conclusion

In this work, we propose a realistic MARL environment for the online order-dispatching and fleet management problems in a food-delivery system. We consider a setting where the drivers are not homogeneous as opposed to the existing approaches in the literature. While solving the order-dispatching task, we consider the personal constraints of drivers such as the location of their homes. We also provide a reward structure to capture the effect of the long distances travelled by the drivers to pick-up the orders from the restaurants both on the long-term customer satisfaction level and on the returns obtained by drivers. Lastly, we build an efficient food-delivery simulator based on our MARL environment and conduct experiments using three different AC algorithms. We find that the highly dynamic and stochastic nature of our environment and the complex interactions between the drivers result in high fluctuations in the performance of the AC-based methods. We discuss some potential causes for this result and aim to perform further analyses to stabilize our framework as a future work.

References

Abubakr O Al-Abbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4714–4727, 2019.

Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Yong Gao, Dan Jiang, and Yan Xu. Optimize taxi driving strategies based on reinforcement learning. *International Journal of Geographical Information Science*, 32(8):1677–1696, 2018.

Jintao Ke, Feng Xiao, Hai Yang, and Jieping Ye. Optimizing online matching for ride-sourcing services with multi-agent deep reinforcement learning. *arXiv preprint arXiv:1902.06228*, 2019.

Minne Li, Zhiwei (Tony) Qin, Yan Jiao, Yaodong Yang, Zhichen Gong, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *Proceedings of the 2019 World Wide Web Conference (WWW'19),*. ACM, New York, NY, USA, 2019. doi: https://doi.org/10.1145/3308558.3313433.

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

Takuma Oda and Carlee Joe-Wong. Movi: A model-free approach to dynamic fleet management. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2708–2716. IEEE, 2018.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A deep value-network based approach for multi-driver order dispatching. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1780–1790, 2019.

Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 617–626. IEEE, 2018.

Jian Wen, Jinhua Zhao, and Patrick Jaillet. Rebalancing shared mobility-on-demand systems: A reinforcement learning approach. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 220–225. IEEE, 2017.

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

Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

Yongjian Yang, Xintao Wang, Yuanbo Xu, and Qiuyang Huang. Multiagent reinforcement learning-based taxi predispatching model to balance taxi supply and demand. *Journal of Advanced Transportation*, 2020, 2020.

Ming Zhou, Jarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. In *Proceedings of The 28th ACM International Conference on Information and Knowledge Management,*, 2019. doi: https://doi.org/10.1145/3357384.3357799.