

# MULTI-AGENT REINFORCEMENT LEARNING FOR DYNAMIC PRICING AND FLEET MANAGEMENT IN AUTONOMOUS MOBILITY-ON-DEMAND SYSTEMS

Arthur Wang      Berkay Turan

Department of Electrical and Computer Engineering

University of California, Santa Barbara

Santa Barbara, CA, 93106

chengyilin@ucsb.edu      bturan@ucsb.edu

Faculty Advisor:

Mahnoosh Alizadeh

## ABSTRACT

This paper considers the joint fleet management and ride pricing problem faced by a profit-maximizing transportation service provider that operates a fleet of autonomous vehicles to serve the population's urban mobility needs. Due to intractability issues of solving for the exact optimal real-time control policy, reinforcement learning-based solutions have been shown to perform well [1]. Existing works based on reinforcement learning do not scale well with the network size due to large state and action spaces. We study this issue by applying multi-agent reinforcement learning to develop a real-time policy, where each node acts as an agent. State abstraction allows us to represent the state information in a more compact manner, which reduces the dimension of the state space, while local decisions result in a lower dimension for the action space. We demonstrate the efficacy of the multi-agent reinforcement learning policy on an 8-node transportation network.

## INTRODUCTION

The rapid evolution of enabling technologies for autonomous driving has facilitated state-of-the-art transportation options for urban mobility. Owing to developments in automation, it is possible for an autonomous-mobility-on-demand (AMoD) fleet of autonomous vehicles to serve the society's transportation needs, with multiple companies now heavily investing in AMoD technology [2].

The introduction of autonomous vehicles for mobility on-demand services provides an opportunity for better fleet management. Specifically, idle vehicles can be *rebalanced* throughout the network in order to prevent accumulating at certain locations and to serve induced demand at every location. Autonomous vehicles allow rebalancing to be performed centrally by a platform operator who observes the state of all the vehicles and the demand, thus eliminating the need for manual intervention from a human driver. Moreover, a dynamic pricing scheme for rides is essential to maximize profits earned by serving the customers. Coupling an optimal fleet management

policy with a dynamic pricing scheme allows the revenues to be maximized while reducing the rebalancing cost and the waiting time of the customers by adjusting the induced demand.

We consider a model that captures the opportunities and challenges of an AMoD fleet, which consists of complex state and action spaces. In particular, the platform operator has to consider the number of customers waiting to be served at each location (ride request queue lengths) and the locations of all the vehicles in order to make decisions. These decisions consist of pricing for rides for every origin-destination (OD) pair and routing decisions for every vehicle in the network. Upon taking an action, the state of the network undergoes a stochastic transition due to the randomness in customer behavior. In this AMoD system, a real-time control policy that jointly executes dynamic pricing and vehicle routing decisions based on the real-time state is required in order to maximize profits and customer satisfaction.

The earlier works on control policies for AMoD systems use Model Predictive Control (MPC) approaches to design real-time control policies for fleet management [3, 4, 5, 6]. The underlying idea is to solve an open-loop optimization problem at each time step to yield a sequence of control actions over a receding horizon, but only execute the first control action. The drawback of these approaches is that they rely on predictions of the future (e.g., future demand) for planning, which might not be accurate. Furthermore, the optimization problems are typically formulated into large-scale linear or integer programming problems, which may not scale well. An alternative approach is to formulate the evolution of the AMoD system as a Markov Decision Process (MDP) and solve for the optimal policy using dynamic programming. Due to intractability issues of solving for the exact optimal real-time control policy using dynamic programming, reinforcement learning-based solutions have been shown to perform well [1, 7, 8, 9]. Among these, the multi-agent approaches model each vehicle as an agent and therefore do not scale well with the number of vehicles [8, 9]. On the other hand, the centralized approaches successfully develop real-time control policies for rebalancing [7] as well as joint dynamic pricing and rebalancing [1], however, these approaches do not scale well with the number of nodes in the network.

In this work, we study the scalability issue by applying multi-agent reinforcement learning to develop a real-time policy, where each node acts as an agent. State abstraction allows us to represent the state information in a more compact manner, which reduces the dimension of the state space, while local decisions result in a lower dimension for the action space. Inspired by [10], we propose a multi-agent reinforcement learning framework for AMoD and learn a multi-agent policy via Proximal Policy Optimization (PPO) [11]. We demonstrate that the multi-agent RL policy converges faster and performs better than the centralized RL policy.

## SYSTEM MODEL AND PROBLEM DEFINITION

The following definitions, which are adopted from [1], describe the system model and problem:

**Network and Demand Models:** We consider a fleet of AMoD vehicles operating within a transportation network characterized by a fully connected graph consisting of  $\mathcal{N} = \{1, \dots, n\}$  nodes that can each serve as a trip origin or destination. We study a discrete-time system with time periods normalized to integral units  $t \in \{0, 1, 2, \dots\}$ . In this discrete-time system, we model the arrival of the potential riders (i.e., the customers willing to travel free of charge) with origin-destination (OD) pair  $(i, j)$  as a Poisson process with an arrival rate of  $\lambda_{ij}(t)$  in period  $t$ , where

$\lambda_{ii}(t) = 0$ . We adopted a price-responsive rider model, where we assume that the riders are heterogeneous in terms of their willingness to pay. In particular, if the price for receiving a ride from node  $i$  to node  $j$  in period  $t$  is set to  $p_{ij}(t)$ , the induced arrival rate for rides from  $i$  to  $j$  is given by  $\Lambda_{ij}(t) = \lambda_{ij}(t)(1 - F(p_{ij}(t)))$ , where  $F(\cdot)$  is the cumulative distribution of riders' willingness to pay with a support of  $[0, p_{\max}]$ . Thus, the number of new ride requests in time period  $t$  is  $A_{ij}(t) \sim \text{Pois}(\Lambda_{ij}(t))$  for OD pair  $(i, j)$ .

**Vehicle Model:** To capture the effect of trip demand and the associated routing (routing also implies rebalancing of the empty vehicles) decisions on the costs associated with operating the fleet (maintenance, mileage, etc.), we assume that each autonomous vehicle in the fleet has a per period operational cost of  $\beta$ . In our discrete-time model, we assume each vehicle takes  $\tau_{ij}$  periods to travel between OD pair  $(i, j)$ . We consider the travel times to be constant and exogenously defined for the time period the policy is developed for, because we assume that the number of AMoD vehicles is much less compared to the rest of the traffic. Also, to consider changing traffic conditions throughout the day, it is possible to train multiple control policies for the different time intervals.

**Ride Hailing Model:** The platform operator dynamically routes the fleet of vehicles in order to serve the demand at each node. Customers that purchase a ride are not immediately matched with a ride but enter the queue for OD pair  $(i, j)$ . After the platform operator executes routing decisions for the fleet, the customers in the queue for OD pair  $(i, j)$  are matched with rides and served in a first-come, first-served discipline. A measure of the expected wait time is not available to each arriving customer. However, the operator knows that longer wait times will negatively affect their business and hence seeks to minimize the total wait time experienced by users. Denote the queue length for OD pair  $(i, j)$  by  $q_{ij}(t)$ . If after serving the customers, the queue length  $q_{ij}(t) > 0$ , the platform operator is penalized by a fixed cost of  $w$  per person at the queue to account for the value of time of the customers.

**Platform Operator's Problem:** We consider a profit-maximizing AMoD operator that manages a fleet of vehicles that make trips to provide transportation services to customers. The operator's goal is to maximize profits by 1) setting prices for rides and hence managing customer demand at each node; 2) optimally operating the AMoD fleet (i.e., routing) to minimize operational costs as well as customer wait time.

## BRIEF OVERVIEW OF DEEP REINFORCEMENT LEARNING

Reinforcement learning is a machine learning approach in which an agent learns to choose optimal actions to achieve its goals by observing the states of its environment through an interactive process. The main purpose of an agent in reinforcement learning is to learn a policy that from any initial state, performs actions that maximize the reward accumulated over time. The general setting of reinforcement learning can be summarized as follows. At an instant  $t$ , An agent observes the environment's current state  $s(t) \in \mathcal{S}$ , and performs an action  $a(t) \in \mathcal{A}$ , where  $\mathcal{S}$  and  $\mathcal{A}$  are state and action spaces of the environment, respectively. The environment responds to the agent's action  $a(t)$  at state  $s(t)$  with a reward  $r(t) = r(s(t), a(t))$  and generates a successor state  $s(t+1) = \mathcal{T}(s(t), a(t))$ , where  $\mathcal{T}$  is the random and unknown transition function

of the environment. In deep reinforcement learning, the goal of the agent is to find a stochastic (or deterministic) policy parameterized by  $\theta$ , which constitute the weights of a deep neural network:  $\pi_\theta(a|s) = \pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ , i.e., a probability distribution in the state-action space (or  $\pi_\theta(s) : \mathcal{S} \rightarrow \mathcal{A}$  in the deterministic case). The goal is to derive the optimal policy  $\pi^*$ , which maximizes the discounted cumulative expected rewards:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r(t) \right], \quad (1)$$

where  $\gamma \in (0, 1]$  is the discount factor. The value of taking an action  $a$  in state  $s$ , and following the policy  $\pi$  afterwards is characterized by the value function  $Q_{\pi}(s, a)$ :

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r(t) | s(0) = s, a(0) = a \right]. \quad (2)$$

The methods used by reinforcement learning algorithms can be divided into three main groups: 1) critic-only methods, 2) actor-only methods, and 3) actor-critic methods, where the word critic refers to the value function and the word actor refers to the policy [12]. In this work, we adopt a practical policy gradient method called Proximal Policy Optimization (PPO) [11] to develop our real-time policy, which is an actor-critic method. In actor-critic methods, the policy is updated by making use of both the value functions and policy gradients:

$$\theta(t+1) = \theta(t) + \alpha \nabla_{\theta} \mathbb{E}_{\pi_{\theta(t)}} \left[ Q_{\pi_{\theta(t)}}(s, a) \right]. \quad (3)$$

Actor-critic methods are able to produce actions in a continuous action space while reducing the high variance of the policy gradients by adding a critic (value function).

## SINGLE AND MULTI-AGENT REINFORCEMENT LEARNING POLICIES

We employ the deep reinforcement learning method described in the previous section to acquire a decision-making model that can produce near-optimal actions in the AMoD environment. We will train and compare the performances of two policies: 1) a centralized policy (single agent), where all decisions are made by a central operator that observes the complete state of the system, and 2) a multi-agent policy, where each node makes its own pricing and vehicle routing decisions by observing only the relevant states to them. The details of both settings are described below:

1. *Single-agent*: In the single-agent setup,  $s(t)$ , i.e., the state of the environment at time  $t$ , consists of the vehicle locations and the customer queue lengths. We let  $s(t) = [v(t) \ q(t)]$ , where  $v(t)$  is the vector that consists of the number of vehicles at each node (as well as the vehicles currently traveling) and  $q(t) = [q_{ij}(t)]_{i,j \in \mathcal{N}}$  is the vector that consists of the customer queue length for all OD pairs  $(i, j)$ . Upon observing the state  $s(t)$  of the AMoD system, the agent takes action  $a(t)$ . We let  $a(t) = [p(t) \ x(t)]$ , where  $p(t) = [p_{ij}(t)]_{i,j \in \mathcal{N}}$  consists of ride prices between all OD pairs  $(i, j)$  and  $x(t) = [x_{ij}(t)]_{i,j \in \mathcal{N}}$  consists of number

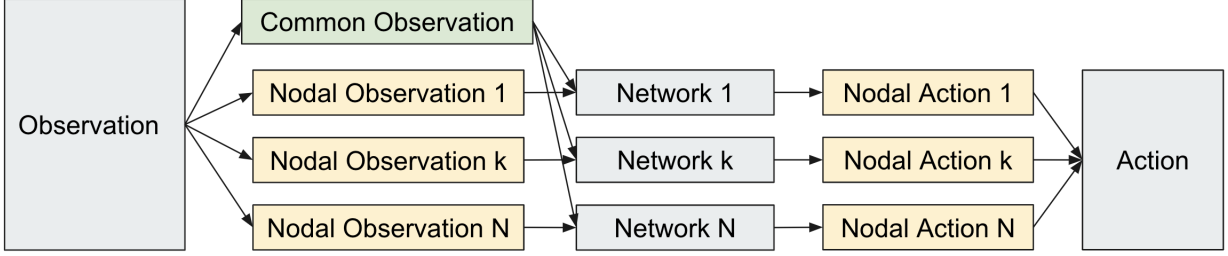


Figure 1: Data Flow of Multi-Agent Training

of vehicles being moved from every origin  $i$  to every destination  $j$ . Upon taking the action  $a(t)$ , the agent observes the reward:

$$r(t) = \sum_{i,j \in \mathcal{N}} p_{ij}(t) A_{ij}(t) - w \sum_{i,j \in \mathcal{N}} q_{ij}(t) - \beta \sum_{i,j \in \mathcal{N}} \tau_{ij} x_{ij}(t), \quad (4)$$

where the first term corresponds to the revenue generated by the passengers that request a ride for a price  $p_{ij}(t)$ , the second term is the queue cost of the passengers that have not yet been served, and the third term is operational costs of the vehicles that are being routed.

2. *Multi-agent*: In the multi-agent setup, the idea is to reduce the dimension of inputs and outputs of the neural network by state abstraction and local decisions, and thus reduce the required size or training time to achieve an accurate result. This is applicable when the action space of the environment can be divided into multiple parts, where each part only requires a fraction of the observation and/or summary of the observation. In the context of the AMoD system, each node has its own observation relevant to its local decisions. We let the state of node  $i$  be  $s_i(t) = [q_{ij}(t) q_{ji}(t) v_j(t) \sum_{k \in \mathcal{N}} q_{jk}(t)]_{j \in \mathcal{N}}$ , which consists of the queues for which node  $i$  is either origin or destination, the available vehicles at each node  $j \in \mathcal{N}$ , and the total queue lengths at every node  $j \in \mathcal{N}$ . We let the action taken by node  $i$  be  $a_i(t) = [x_{ij}(t) p_{ij}(t)]_{j \in \mathcal{N}}$ , which consists of vehicles sent to each destination  $j$  and prices set for each ride origination from  $i$ . Once all nodes take their local actions, the reward is observed the same as (4).

Figure 1 illustrates the data flow of multi-agent architecture. It is worthwhile to highlight that while the state and the action spaces of the single-agent setup are  $\mathcal{O}(n^2)$ , the state and the action spaces of the multi-agent setup are  $\mathcal{O}(n)$ . This reduction in the space dimensions motivates the multi-agent approach for faster training of the neural networks. In the next section, we present the numerical study comparing the performances of the single and the multi-agent policies.

## NUMERICAL RESULTS

We build the AMoD environment using OpenAI-Gym [13], and use Proximal Policy Optimization (PPO) [11] on Stable Baselines 3 [14] to train a decision making model for this problem. The value network and policy network of PPO are both fully connected feed-forward layers with ReLU activation functions.

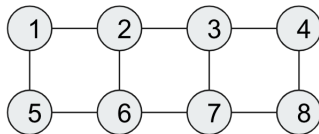


Figure 2: Map of Testing Environment

For both single and multi-agent setups, we used a single neural network for the policy. Although it is straightforward to have a single neural network for the single-agent setup, some justification is required for the multi-agent setup. Theoretically, each agent of the multi-agent learning model should have its own policy, i.e., independent networks. However, in the context of AMoD environment, the network weights should be very similar. The only difference is that the difference in distance to each other node causes the policy to have slightly different weights. Treating the distances as part of the input to each sub-networks, we can use one-hot encoding of the current node index as the input, and the weight of this input will store the distance information. By doing so, for an  $n$  node environment, we need only 1 network instead of  $n$  networks, increasing the training speed significantly.

**Testing Environment:** As illustrated in Figure 2, the sample environment consists of 8 nodes arranged in a 2 by 4 grid, where each node is connected to adjacent nodes with a distance of 1. Thus, the furthest node pair has a distance of 4 time units. The arrival rate  $\lambda_{ij} = \bar{\lambda}\tau_{ij}^{-1}$ , where  $\tau_{ij}$  is the distance (in time units) between node  $i$  and node  $j$ . In this model, there will be fewer customers on longer trips.

**Environment Parameters:** We let  $p_{ij}(t) \in [0, 1]$  and  $F(p_{ij}(t)) = 1 - p_{ij}(t)$ . We use the arrival rate  $\bar{\lambda} = 5$ , operating cost  $\beta = 0.1$ , waiting penalty  $w = 0.2$ , and total number of vehicles  $N_v = 80$ . This number is chosen deliberately small so that stationary policy cannot perform well: When there are more than enough vehicles available, a response-on-demand policy is already optimal, while having fewer vehicles will make it harder to have a stable policy.

**Training Parameters:** We adapt the default parameters of Stable Baselines 3 for single-agent (subscript  $s$  below), but did some changes to fit the one-network multi-agent model (subscript  $m$  below):

- Discount factor  $\gamma_s = \bar{\gamma} = 0.99$ ,  $\gamma_m = \bar{\gamma}^{1/n} = 0.99^{1/8} = 0.99874$
- GAE parameter  $\lambda_s = \bar{\lambda} = 0.95$ ,  $\lambda_m = \bar{\lambda}^{1/n} = 0.95^{1/8} = 0.9936$
- Sampling Step  $n_{\text{step},s} = \bar{n}_{\text{step}} = 256$ ,  $n_{\text{step},m} = \bar{n}_{\text{step}}n = 256 \times 8 = 2048$
- Size of value network and policy network: 2 hidden layers of size 128 neurons.
- Learning rates:  $1 \times 10^{-5}$ ,  $2 \times 10^{-5}$ ,  $3 \times 10^{-5}$

The training results are shown in Figure 3. It can be seen that the single-agent results are not improving when reducing the learning rate, which means the bottleneck of improvement is on the network size. For single-agent, there are 80 observations and 128 actions, which probably require a network far larger than 128 neurons per layer. On the other hand, multi-agent not only converges to a far better result but also shows an improving trend when reducing learning rates. This result

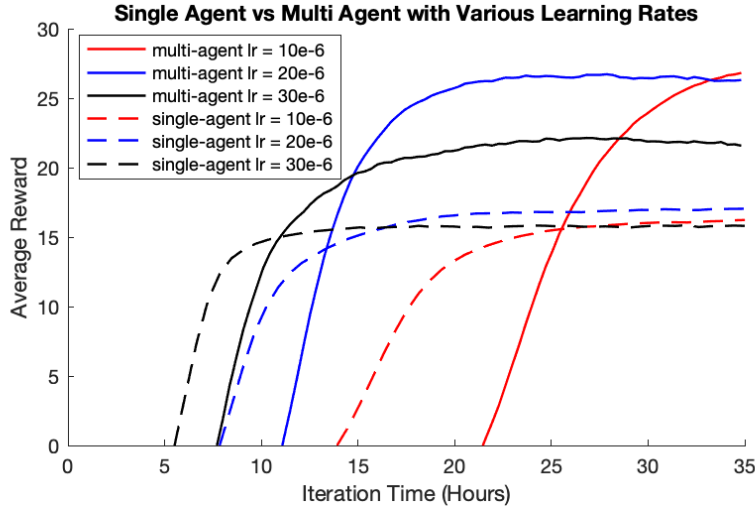


Figure 3: Training Results

shows that multi-agent is an effective way to reduce network size requirements and improve the result.

## CONCLUSIONS

In this paper, we developed a real-time control policy based on deep reinforcement learning for operating an AMoD fleet of vehicles as well as pricing for rides. Our real-time control policy jointly makes decisions for: 1) vehicle routing in order to serve passenger demand and rebalance the empty vehicles, and 2) pricing for rides in order to adjust the potential demand so that the network is stable and the profits are maximized. We developed two real-time policies based on reinforcement learning: 1) a single-agent policy, where all decisions are made by a central agent that observes the complete state of the system, and 2) a multi-agent policy, where each node makes its own pricing and vehicle routing decisions by observing only the relevant states to them. Although the single-agent reinforcement learning policy performs well for small networks, it does not scale well with the number of nodes in the network. On the other hand, the multi-agent reinforcement learning policy suffers less from the number of nodes thanks to state abstraction and local decisions. Through a numerical study on an 8-node network, we demonstrate the superiority of the multi-agent policy to the single-agent policy.

## ACKNOWLEDGEMENTS

This work is supported by the NSF Grant #1837125 and the International Foundation for Telemetering (IFT).

## REFERENCES

- [1] B. Turan, R. Pedarsani, and M. Alizadeh, “Dynamic pricing and fleet management for electric autonomous mobility on demand systems,” *Transportation Research Part C: Emerging Technologies*, vol. 121, p. 102829, 2020.
- [2] [Online]. Available: <https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/>.
- [3] R. Zhang, F. Rossi, and M. Pavone, “Model predictive control of autonomous mobility-on-demand systems,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1382–1389, IEEE, 2016.
- [4] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas, “Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 463–478, 2016.
- [5] R. Iglesias, F. Rossi, K. Wang, D. Hallac, J. Leskovec, and M. Pavone, “Data-driven model predictive control of autonomous mobility-on-demand systems,” in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6019–6025, IEEE, 2018.
- [6] M. Tsao, R. Iglesias, and M. Pavone, “Stochastic model predictive control for autonomous mobility on demand,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3941–3948, IEEE, 2018.
- [7] D. Gammelli, K. Yang, J. Harrison, F. Rodrigues, F. C. Pereira, and M. Pavone, “Graph neural network reinforcement learning for autonomous mobility-on-demand systems,” *arXiv preprint arXiv:2104.11434*, 2021.
- [8] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye, “Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem,” in *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 1090–1095, IEEE, 2019.
- [9] M. Guériau, F. Cugurullo, R. A. Acheampong, and I. Dusparic, “Shared autonomous mobility on demand: A learning-based approach and its performance in the presence of traffic congestion,” *IEEE Intelligent Transportation Systems Magazine*, vol. 12, no. 4, pp. 208–218, 2020.
- [10] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Advances in neural information processing systems*, vol. 30, 2017.
- [11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [12] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

- [13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” 2016.
- [14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.