Deep Reinforcement Learning-based Trajectory Pricing on Ride-hailing Platforms

JIANBIN HUANG, LONGJI HUANG, MEIJUAN LIU, HE LI, QINGLIN TAN, XIAOKE MA, and JIANGTAO CUI, Xidian University, China
DE-SHUANG HUANG, Guangxi Academy of Science, China and Tongji University, China

Dynamic pricing plays an important role in solving the problems such as traffic load reduction, congestion control, and revenue improvement. Efficient dynamic pricing strategies can increase capacity utilization, total revenue of service providers, and the satisfaction of both passengers and drivers. Many proposed dynamic pricing technologies focus on short-term optimization and face poor scalability in modeling long-term goals for the limitations of solution optimality and prohibitive computation. In this article, a deep reinforcement learning framework is proposed to tackle the dynamic pricing problem for ride-hailing platforms. A soft actor-critic (SAC) algorithm is adopted in the reinforcement learning framework. First, the dynamic pricing problem is translated into a **Markov Decision Process (MDP)** and is set up in continuous action spaces, which is no need for the discretization of action space. Then, a new reward function is obtained by the order response rate and the KL-divergence between supply distribution and demand distribution. Experiments and case studies demonstrate that the proposed method outperforms the baselines in terms of order response rate and total revenue.

CCS Concepts: • **Information systems** → **Data mining**; • **Computing methodologies** → **Machine learning;**

Additional Key Words and Phrases: Trajectory dynamic pricing, traffic management, Reinforcement learning

ACM Reference format:
Jianbin Huang, Longji Huang, Meijuan Liu, He Li, Qinglin Tan, Xiaoke Ma, Jiangtao Cui, and De-Shuang Huang. 2022. Deep Reinforcement Learning-based Trajectory Pricing on Ride-hailing Platforms. *ACM Trans. Intell. Syst. Technol.* 13, 3, Article 41 (March 2022), 19 pages.
https://doi.org/10.1145/3474841

1 INTRODUCTION

Ride-hailing platforms such as Uber [4] and Didi Chuxing [1] have experienced explosive growth in recent years and have substantially transformed our lives. The platforms match passengers who

This work is supported in part by the National Science Foundation of China (Grant Number: 61876138). Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.
Authors' addresses: J. Huang, L. Huang (corresponding author), M. Liu, H. Li (corresponding author), Q. Tan, X. Ma, and J. Cui, Xidian University, No. 2 South Taibai Road, XiAn, Shanxi, China, 710126; emails: jbhuang@xidian.edu.cn, {longji.huang018, meyonliu}@gmail.com, heli@xidian.edu.cn, qltan@stu.xidian.edu.cn, {xkma, cuijt}@xidian.edu.cn; D.-S. Huang, Guangxi Academy of Science, No. 1239, Siping Road, Yangpu District, Nanning, GuangXi, China, 200092, Tongji University, No. 1239, Siping Road, Yangpu District, Shanghai, China, 201804; email: dshuang@tongji.edu.cn.

2157-6904/2022/03-ART41 $15.00
https://doi.org/10.1145/3474841

request a ride in real-time with nearby idle vehicles and then charge a part of the payment as its revenue after finishing the trip. Therefore, the platforms need to leverage various levers to promote order dispatch efficiency and improve utilization of transportation resources and thus increase the total revenue.

One of the key challenges for improving order dispatch efficiency in the ride-sharing platform is to better balance the demands and supplies. On the one hand, the spatio-temporal mismatch scenario between vehicles and passengers is common in the real world, such as the cases of oversupply at off-peak hours and excessive demand at peak hours. In particular, when an order dispatch system is over-burdened under oversupply and excessive demand, vehicles will have to be dispatched on a wild goose chase to pick up a rider at a distant location. As a result, drivers will spend a long time picking up their passengers and thus taxis will idle infrequently, which will reinforce the scarcity of idle drivers and inefficiency of transportation [9]. On the other hand, the current strategies of the platform will significantly impact the future geographic distribution of drivers. If the platform makes decisions with long term reward, guiding more available drivers to to pick up passengers in locations with high demand, which greatly increases the number of orders being served in platform. To solve the challenge above, many measures are proposed to balance the demand and supply, dynamic pricing is one of the popular measures. Dynamic pricing adjusts the price for rides based on real-time demand and supply conditions, which gives benefits on peak load balance, order response rate improvement, and revenue maximization.

Generally, both drivers and passengers are sensitive to price, lower prices will encourage more passengers to pay for their trips while the higher price will motivate more drivers to provide service. The ride-hailing platforms just exploit this property to make price a powerful tool to achieve revenue management. For example, Uber and Lyft have implemented surge pricing on their platforms; when demand is high relative to supply, the base fare is multiplied by a multiplier that is greater than one. Surge pricing can significantly motivate drivers to work more time at high surge times [10] and it can efficiently solve the wild goose chase [9]. However, due to most of the scenarios start with high demand and low supply, passengers usually pay more for their trip, and surge pricing only balances supply and demand for the current period and does not consider the future condition. Some dynamic pricing algorithms consider short-term supply and demand forecasts [5, 33]. For example, Reference [5] gives consideration to the future demand and supply condition by guiding more drivers to locations with high demand in the future; it generates more revenue for the platform and provides lower price for passengers. However, limitations exist in terms of scalability and expensive computational cost when it extends to consider revenue maximization for a longer horizon. **Reinforcement learning (RL)** has the ability to address this challenge, thus there are many pricing methods based on RL [14, 17, 20, 28, 31]. The agent gradually learns how to improve its pricing strategies (actions) by utilizing experiences collected from interactions with the environment. However, they usually set up the dynamic pricing problem in discrete action spaces and solve it with conventional Q-learning algorithms, which suffers obviously from difficulty setting the proper number of discrete actions. If a small number of discrete actions are considered, then different prices in a large pricing area will be regarded as the same price for the agent, the feedback the agent receives regarding the impact of pricing actions is imprecise, leading to sub-optimal decisions. Meanwhile, as the number of discrete actions increases, the dimensional curse and the computational burden will suffer. Besides, these methods are not designed for ride-hailing platforms and its objectives are different from the ride-hailing platforms. Reference [34] shows that a higher-order response rate usually means higher revenue, thus the ride-hailing platform maximizes the order response rate and total revenue at the same time. Therefore, precious methods based on RL can not be directly extended to the ride-hailing platform.

In this article, we propose a dynamic pricing method for the ride-hailing platform, aiming to maximize the long term **accumulative platform revenue (APR)** and the **order response rate (ORR)**. RL naturally fits the problem of maximizing the long-term rewards. Deep reinforcement learning combines RL and deep neural network [25] to approximate the action-value function, which easily maps continuous states to continuous actions, avoiding the discretization of action spaces. Thus, we employ DRL to solve dynamic pricing for ride-hailing platforms. To find long-term strategies to improve long-term APR and ORR, it is necessary to take the future demand and supply into consideration when making decisions.

There are some inevitable challenges needed to tackle. First, the reward function should be carefully designed, otherwise leading to non-convergence policies. Second, how to control the future distribution of available drivers is of great significance, because different scheduling policies will lead to a different distribution of drivers, making the objective function sub-optimal.

To further address the challenges above, a new reward function is designed and scheduling is jointly optimized with pricing. First, the reward function maximizes APR and ORR, which consists of three terms (including the instantaneous revenue, the order response rate that reflects the service quality, and the KL-divergence that reflects the difference between demand distribution and supply distribution). Second, if dynamic pricing is utilized as the single marketplace lever, the price will be high and volatile [33]. Besides, optimization of the only price does not optimize the overall system and is unfavorable for controlling the future distribution of vehicles. Hence, scheduling and pricing will be jointly optimized. Finally, we set up the dynamic pricing problem in multi-dimensional continuous action spaces, and the agent needs to make an efficient tradeoff between exploration and exploitation. Thus, we employ **Soft Actor-Critic (SAC)**, [13] which enables stability and full exploration by maximizing expected reward function and entropy of policies, to find the optimal pricing and scheduling policies. More specifically, The contribution of this work is summarized as follows:

- A deep reinforcement learning framework is proposed to solve the dynamic pricing problems for the ride-hailing platform. The problem is set in continuous action spaces and uses the Soft Actor-Critic algorithm to find the optimal policy.
- A new reward function is designed, which consists of three terms, including instant revenue, order response rate, and the KL-divergence between supply distribution and demand distribution, aiming to improve the long-term APR and ORR.
- Experiment and case studies are conducted on real-world datasets to evaluate the performance of the proposed methods. Comparing to the methods based on traditional mathematical optimization, our method achieves higher APR and ORR. Moreover, the proposed reward function helps our model to obtain faster convergence and better performance compared to the RL methods with only revenue as the reward function.

The rest of the article is organized as follows: In Section 2, we review some related literature about dynamic pricing. In Section 3, a Markov decision process of the dynamic pricing problem and a newly designed reward function are detailed. Section 4 details the Soft Actor-Critic algorithm and the overall model framework. We then describe the comparison experiments in Section 5 and give the conclusions in Section 6.

2 RELATED WORK

The two main research areas relevant to this study are dynamic pricing and reinforcement learning; we will briefly describe each of them as follows:

Dynamic pricing. Dynamic pricing is the main strategy for price-based revenue management and thus is nowadays widely applied in many contexts, such as retail and e-commerce [16],

sponsored search auctions [28], intelligent transportation system [6], and so on. There exist different pricing mechanisms for different contexts, such as static pricing, threshold pricing, markdown pricing, discounts, auctions, price negotiations, and so on. In most of the scenarios, the price needs to be varied with respect to the market fluctuations; the context of the ride-hailing market is also not the exception. In specialty, both passengers and drivers are customers of the ride-hailing platform, thus fluctuations and uncertainty exist in both demand and supply. There are many dynamic pricing techniques concerning how to yield more profits for ride-hailing platforms [6, 8, 11, 12, 21], and some popular pricing schemes such as surge pricing are employed by ride-hailing platforms such as Uber and Lyft [23, 33]. Surge pricing outperforms static pricing in revenue maximization [6] and incentivizes drivers to work longer on the road [10] and solves the wild goose chase [9]. However, surge pricing does not consider its effects on customers. When demand exceeds supply, most passengers will be charged an expensive fare due to their trips for the base fare being multiplied by a surge multiplier that is greater than 1. Reference [7] proposes to predict the surge multiplier in the next few minutes to hours to help save passengers' money. Reference [5] presents an adaptive pricing scheme that incorporates the forecast demand and guides more drivers to those regions with high demand and low supply to improve the revenue of drivers and reduce the average payment of passengers. However, these techniques only consider short-term benefits, and there is still more room for improvement by considering longer-term (next few hours or one day) benefits. Moreover, these models will face poor scalability and expensive computational cost when extending to optimize revenue over a longer horizon.

Deep reinforcement learning. Reinforcement learning concerns how an agent takes actions in an environment to achieve a given goal [29]. RL commits to achieve long-term rewards, thus naturally fits for optimizing the problems with long-term goals. Some popular RL algorithms, such as Q-learning [32] and SARSA [24], are often applied to those problems with discrete state and action spaces. Deep RL algorithms approximate the action policy function and critic value function using neural networks, such as DQN [19]. The general actor-critic-based policy gradient algorithms [13, 15, 26, 27] are suited for the continuous state and action spaces. One of the challenges of deep RL is how to make a tradeoff between exploration and exploitation. The agent should not only exploit what it has already known to obtain a reward, but also fully explore the environment to optimize its policy. There are several methods proposed to address this problem, including ϵ-greedy policy for action selection, action noise for deterministic policy gradient algorithms, and entropy regularization. In this article, we use SAC to find the optimal pricing and scheduling policy. Entropy regularization is adopted in SAC for making a tradeoff between expected return and exploration, which uses the entropy of policy as a measure of policy exploitation. Besides, SAC automatically tunes the temperature parameter of entropy regularization during the training process, avoiding the need for manually tuning. Meanwhile, it also can be used in continuous action spaces, suiting for dynamically continuous pricing problems.

RL-based dynamic pricing. There are many existing agent-based approaches to the dynamic pricing problem. References [14, 17] utilize reinforcement learning to optimize price for the energy market. Reference [22] optimizes revenue in a model-free environment and optimizes pricing decisions using the Monte Carlo algorithm. Reference [18] maximizes price while keeping the balance between revenue and fairness using Q-learning. However, these solutions are designed for discrete prices; it is more practical with a continuous price in the ride-hailing market. Besides, these previous works taking revenue as the reward function work out well, but not in the ride-hailing market. So, those methods can not be directly applied in the dynamic pricing tasks for the ride-hailing platform.

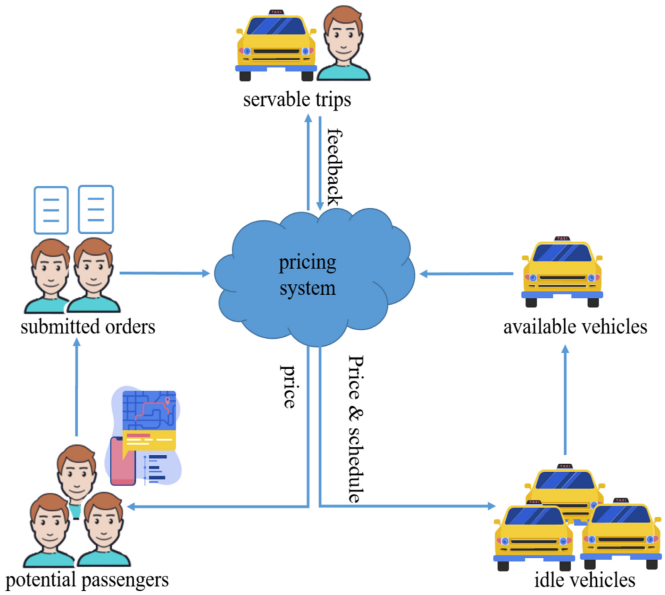


Fig. 1. Illustration of dynamic pricing process in ride-hailing systems.

3 METHODOLOGY

In this section, we first give the preliminary of pricing problem and then formulate the dynamic pricing problem as a **Markov Decision Process(MDP)**, giving the description of the state, action, reward function, and state transition.

3.1 Preliminary

We characterize the transportation network using a fully connected graph consisting of N nodes, each node can be acted as a trip origin or destination and contains available drivers and potential passengers. The set of all nodes is given by $G = \{v_i | i \in 1, 2, \ldots, N\}$. The duration of one day is split into T time-steps, and the distance between two time-steps is defined by Δt. At each time-step, the potential passengers emerge in each node and may request a ride. Those rides are served by vehicles, which are available in the same node and willing to serve. Such a process will be formalized more concretely as following.

As shown in Figure 1, when a user opens the ride-hailing application on the phone and selects his/her origin and destination, before submitting a ride request, the user becomes a potential demand. At each time-step t, there are potential demands with origin i and destination j, which is denoted by C_{ij}^t. Every user has a reservation value for ride fare that influences the user's behaviors. If the ride's price given by the platform is lower than the reservation value, then he/she will submit a ride request, otherwise will quit the platform. As shown in this process, there exists a relationship between price and demand, describing how passengers respond to different rides' prices. We assume passengers are homogeneous in terms of their willingness to pay, and there exists a function $F^r(p)$, which represents the relationship between demand and price. This function gives the cumulative distribution of riders' willingness to pay at price p, $p \in [0, p_{max}]$. Finally, the number of passengers who are willing to pay a ride from i to j at time t can be represented as $D_{ij}^t(p_{ij}^t)$; the formulation is defined as follows:

$$D_{ij}^t\left(p_{ij}^t\right) = C_{ij}^t\left(1 - F^r\left(p_{ij}^t\right)\right). \tag{1}$$

The definition of demand (passengers) is shown above, that of supply (drivers) is similar. Available drivers at region i are represented as V_i^t and a fraction b_{ij} of available drivers are scheduled to service the rides from i to j, $\sum_j b_{ij} = 1$. Similar to passengers, drivers' willingness to participate in the platform is also influenced by price, which can be drawn from a distribution $F^v(p)$. So, the network's supply from i to j can be computed as follows:

$$U_{ij}^t\left(p_{ij}^t\right) = V_i^t b_{ij}^t F^v\left(p_{ij}^t\right). \tag{2}$$

Given the demand $D_{ij}^t(p_{ij}^t)$ and supply $U_{ij}^t(p_{ij}^t)$ from i to j at time t, the total number of successfully matched orders can be computed as follows:

$$O_{ij}^t\left(p_{ij}^t\right) = min\left(D_{ij}^t\left(p_{ij}^t\right), U_{ij}^t\left(p_{ij}^t\right)\right). \tag{3}$$

Assume the ride-hailing platform retains a fraction (λ) of the fare paid by passengers as its service fee and gives the rest to the driver, then the platform's total revenue generated at time t would be

$$Rev^t = \lambda \sum_i \sum_j p_{ij}^t O_{ij}^t\left(p_{ij}^t\right). \tag{4}$$

The main purpose of the platform is to maximize the total revenue from t to T.

$$maxRev_{t:T} = max \sum_t^T Rev^t. \tag{5}$$

Most previous work often takes the price as the only parameter, while treats the scheduling fraction of vehicles as a constant term, optimizing the revenue of each timestep independently. Because the drivers' movement in the network significantly affects the future supply in different regions, it is myopia to maximize short-term revenue without considering future demand and supply conditions. Therefore, the platform should consider a long-term revenue optimization by scheduling more drivers to the locations with a large demand-supply gap in the future, which means that the platform needs to optimize price and schedule jointly.

3.2 Formulating Dynamic Pricing as a Markov Decision Process

In this section, the dynamic pricing using reinforcement learning is formulated, and the goal of the agent is to maximize the long-term APR and ORR. We model the problem as a **Markov decision process (MDP)**; the MDP framework can be formulated as a four-tuple $\langle S, A, R, P\rangle$, where S is the state space, A is the action space, R represents the reward function, and P describes the state-action transition operator. The definitions of each element are shown as follows:

3.2.1 State S. The state space contains two elements, $S(t) = \langle C(t), S_v(t)\rangle$, where $C(t) = [C_{ij}^t]_{i,j\in N}$ is the potential demand matrix with size $N \times N$, in which each element C_{ij}^t represents the number of potential passengers from region i to region j at time t. $S_v(t) = \{\langle i, j, \tau\rangle\}$ maintains the states of vehicles, the triple $\langle i, j, \tau\rangle$ represents the status of a vehicle servicing the trip from i to j, τ is the remaining time-steps from the destination. Different rides have different duration times, so different vehicles have different states. Some drivers are busy serving passengers, while other drivers remain idle or become available again after finishing a trip. For example, the drivers at the state $\langle i, i, 0\rangle$ indicate that they remain idle and do not take any order, while the drivers at the state $\langle i, j, \tau\rangle$ indicate that they are on the road to passengers' destination. If a driver finishes a trip, then the state will be updated by $\langle j, j, 0\rangle$.

3.2.2 Action A. The action for dynamic pricing tasks in most contexts is usually the price. But for the ride-hailing platform, utilizing price as the single marketplace lever, the price can be high and volatile. So, the platform needs to guide more vehicles to locations with high demand in the future. Therefore, the action spaces consist of prices of rides at each origin-destination pair and schedule faction of available drivers at each region, which is expressed as $A = \langle P(t), B(t) \rangle$. The first term is $P(t) = [p_{ij}^t]_{i,j \in N}$, where p_{ij}^t is the price for the trip from i to j at time-step t, and is continuous in range $[0, p_{max}]$. The second term is $B(t) = [b_{ij}^t]_{i,j \in N}$, where b_{ij}^t represents the schedule fraction of available drivers from region i to region j, subjecting to the constraint of $\sum_j \beta_{ij}^t = 1$. The distribution of drivers in current and next time-steps can be controlled by using this action, facilitating a better match between drivers and passengers.

3.2.3 State Transition P. Given the action $A(t+1)$ in the current state $S(t)$, the orders emerge in each node and are served by drivers who are available and willing to serve. The drivers at the state $\langle i, i, 0 \rangle$ will transition into state $\langle i, j, \tau_{ij} \rangle$ with probability $b_{ij}^t f^v(p_{ij}^t)$, and transition into the state $\langle i, i, 0 \rangle$ with probability $b_{ij}^t(1 - f^v(p_{ij}^t))$, which means rejecting to provide service and remaining available in place. $f^v(p)$ is the derivative of $F^v(p)$ with respect to price, which indicates the drivers' willingness to serve the trips with price p. The drivers at the state $\langle i, j, \tau_{ij} \rangle$ will transition into the state $\langle i, j, \tau_{ij} - 1 \rangle$ with probability 1 if $\tau_{ij} > 1$, which indicates that the drivers still need to take $\tau_{ij} - 1$ time-steps to their destination, or $\langle j, j, 0 \rangle$ with probability 1 if $\tau_{ij} = 1$, which means the driver finishes his/her trip and becomes available again at region j. By recording the status of each driver, we can compute the number of idle drivers V_i^{t+1} at region i at next time-step as follows:

$$V_i^{t+1} = \alpha_i^{t+1} \left(V_i^t - \sum_j O_{ij}^t + \sum_j I\left(\tau_{ji}^t = 1\right) \right), \tag{6}$$

$$\alpha_i^{t+1} = \frac{\sum_i \sum_j C_{ij}^{t+1}}{\sum_i \sum_j C_{ij}^t}, \tag{7}$$

where $\sum_j O_{ij}^t$ denotes the number of the drivers who take passengers leaving the region i, $I(.)$ is an indicator function, it equals one in the case of $\tau_{ji}^t = 1$ or equals 0 in other cases. $\sum_j I(\tau_{ji}^t = 1)$ denotes the number of the drivers who take riders arriving in the region i and become available again. Notice that the number of drivers keeps changing each time, new vehicles will participate in the platform and existing vehicles will become offline via a random process. Such a random process mainly adjust by multiplying a ratio α_i^{t+1}. α_i^{t+1} represents the ratio between the total number of available drivers and the total number of potential passengers. The definition of α_i^{t+1} can keep the environment relatively stable at each time-step, because the demand-supply pattern remains the same during the whole episode.

For a potential passenger, if the current price is lower than the passengers' reservation value, then he/she will request a ride and be matched to an available driver who is willing to provide service. However, if no available drivers are present, then the passenger will exit the system. If the price is higher than the reservation value, the passenger will also exit the system. Therefore, the potential passengers who are unmatched or do not request a ride will not remain in the system. Thus, for the demand of next time-step, we directly collect the new potential demands of different regions to get the $C(t+1)$, then update the state as $S(t+1) = \langle C(t+1), S_v(t+1) \rangle$.

3.2.4 Reward R. The reward function is the key of reinforcement learning to obtain the right direction of optimization and facilitate the convergence of the training process. It is common that the goal of dynamic pricing is to maximize revenue, thus previous reinforcement learning-based

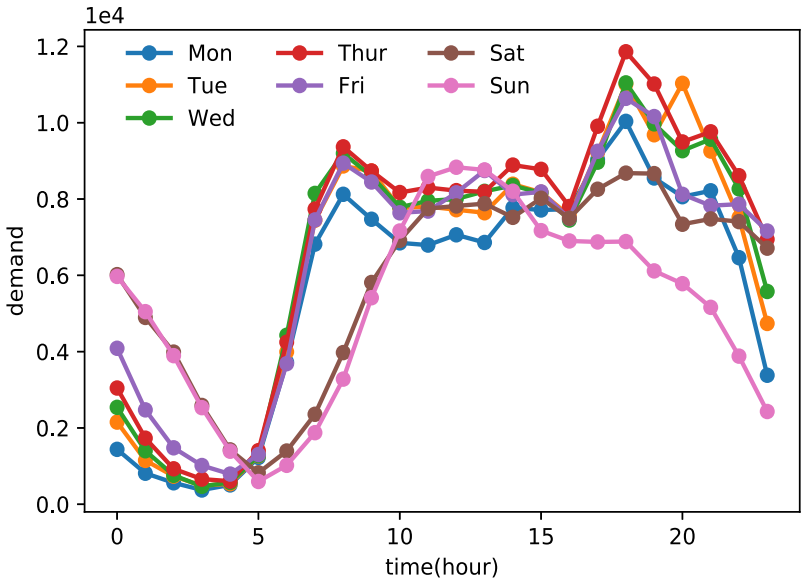


Fig. 2. The number of customers for each day of a week.

techniques usually use revenue as the reward function, which is defined as follows:

$$R_t = Rev(t), \tag{8}$$

where $Rev(t)$ represents the total revenue at time t and can be computed by Equation (4).

However, in the scenario of ridesharing, revenue is determined by price and demand together. The demand is not steady, fluctuating over time. As shown in Figure 2, it is the visualization of the customers' demand for each hour of a day using the yellow taxi data from the New York City Taxi and Limousine Commission dataset [3] for the second week of May 2019. It is obvious that the number of passengers varies greatly from hour to hour, especially before and after rush hours; the total revenue would be dominated by demand in this case. Thus, directly using revenue as a reward function may not work out well to achieve our goals. Therefore, we consider that the reward function also includes the order response rate in addition to the revenue.

The traditional definition of order response rate can be expressed as follows:

$$ORR(t) = \frac{O_t}{C_t}, \tag{9}$$

where O_t represents the total number of final served orders at time-step t and C_t is the total number of potential demands. However, $ORR(t)$ is a small value in range $[0, 1]$ and can be negligible in relation to $Rev(t)$. To narrow the gap between $Rev(t)$ and $ORR(t)$, we redefine the ORR using KL divergence from distribution $O(t)$ to $C(t)$. $C(t)$ is the potential demand matrix. The redefined order response rate would be

$$ORR(t) = D_{KL}(O(t)||C(t)), \tag{10}$$

where $D_{KL}(\cdot)$ represents the KL divergence. According to the property of KL divergence, when the gap between distribution $O(t)$ and $C(t)$ is large, the value of KL divergence is a large negative number. This term will prompt the agent to improve order response rate to obtain high reward value.

Reference [5] shows that guiding more drivers to regions with high future demand reduces the average price and generates more revenue. The insight of this is that a better match between

demand and supply can further improve order response rate and revenue. Therefore, to better match the spatial distribution of vehicles with the distribution of potential demand, we compute the KL divergence from distribution $V(t+1)$ to $C(t+1)$. This KL divergence is denoted as **supply and demand rate (SDR)**.

$$SDR(t+1) = D_{KL}(V(t+1) \| C(t+1)). \tag{11}$$

Finally, the reward function used in our method would be

$$R_t = Rev(t) + \lambda_1 ORR(t) + \lambda_2 SDR(t+1), \tag{12}$$

where λ_1 and λ_2 represent the contribution of KL items. In this setting, the special case of $\lambda_1 = 0$ and $\lambda_2 = 0$ is only take revenue as the reward function.

4 REINFORCEMENT LEARNING MODEL FOR DYNAMIC PRICING

In this section, we formulate the dynamic pricing problem and briefly introduce the basic idea of reinforcement learning algorithm adopted to achieve the optimal price and schedule policies, then detail the main framework of our model.

4.1 Reinforcement Learning for Pricing

Reinforcement learning is a kind of method learning how to map states to actions and maximize a numerical reward signal [29]. At each time-step t, the agent observes current state s_t and then selects an action a_t following its policy π. Next, the environment is translated into a new state s_{t+1} by executing the action a_t and gives the agent a reward $r(s_t, a_t)$. Then the agent will adjust its policy according to the feedback and learn how to maximize the expected cumulative reward over long run. Thus, the main objective of reinforcement learning is to maximize the reward, which can be written as

$$J_\pi = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \pi}[\gamma^t r(s_t, a_t)], \tag{13}$$

where $\gamma \in [0, 1]$ is the discount factor, determining the value of future rewards. The optimal policy can be defined as

$$\pi^* = \arg\max_\pi \mathrm{J}_\pi. \tag{14}$$

Many algorithms can be used to find the optimal policy. On the one hand, there have Q-table methods like Q learning, which obtains the optimal policy by searching the table that stores Q values of all state-action pairs. It is suitable for problems with low dimensional state-action space. On the other hand, there are DRL algorithms such as TRPO, DDPG, and SAC, which use neural networks to represent the state value, state-action value, and policy. It is suitable for the problems with high-dimensional state-action space.

The state and action space of the dynamic pricing problem are continuous, thus DRL algorithms can be used to solve the problem. Because both dimensions increase with the number of regions divided, the DRL algorithms adopted should encourage full exploration. Among the DRL algorithms mentioned above, TRPO is on-policy learning that the collected samples are used only once to update the policy, which faces poor sample efficiency. DDPG and SAC are off-policy learning that the collected samples can be reused many times to update the policy. However, DDPG will struggle to obtain good results in high-dimensional tasks, and it suffers from brittleness due to the temperature hyperparameter of action noise, while SAC has three key advantages compared to the other two DRL methods. First, it is an off-policy framework that reuses experience for data efficiency. Second, it maximizes expected reward function and entropy to enable stability and exploration.

Third, it can automatically tune hyperparameters to avoid the trouble of manually tuning parameters. Therefore, SAC is leveraged to solve the dynamic pricing problem.

4.2 Soft Actor-Critic Algorithm for Pricing

To encourage the agent to explore the environment fully, SAC maximizes the expected reward while also maximizes entropy. So, the optimal policy should be

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))]. \tag{15}$$

The temperature parameter α determines the relative importance of entropy term against the reward, controlling the stochasticity of policy.

SAC belongs to actor-critic based method, so both actor networks and critic networks need to be trained concurrently. We choose three fully connected layers to approximate the policy π_θ, Q-function Q_ϕ, and value function V_φ; the parameters of these networks are denoted by θ, ϕ, and φ, respectively. The weights of the actor and critic networks are randomly initialized and then recursively updated using the transitions (s_t, a_t, r_t, s_{t+1}) sampled from experience replay pool. We will next derive update rules for these parameters.

For the V-network, it takes the environment state s_t as input and outputs the soft state value function $V_\varphi(s_t)$, and it can be trained by minimizing the squared residual error

$$J_V(\varphi) = \mathbb{E}_{s_t\sim D}\left[\frac{1}{2}(V_\varphi(s_t) - \hat{V}(s_t))^2\right], \tag{16}$$

where D is the experience replay pool, and the target soft state value function can be evaluated by

$$\hat{V}(s_t) = \mathbb{E}_{a_t\sim\pi_\theta}\left[Q_\phi(s_t,a_t) - \alpha log\pi_\theta(a_t|s_t)\right]. \tag{17}$$

For the Q-network, it takes the environment state s_t and the action a_t following the policy π_θ as input and outputs the soft Q value $Q_\phi(s_t, a_t)$. The parameters of the Q-network can be updated by minimizing the soft Bellman residual

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}(Q_\phi(s_t,a_t) - \hat{Q}(s_t,a_t))^2\right], \tag{18}$$

with

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[V_{\bar{\varphi}}(s_{t+1})], \tag{19}$$

where the value function can be obtained by a target value network with parameters $\bar{\varphi}$.

For the policy network, it takes the environment state s_t as input and outputs the continuous action $a_t = \pi_\theta(s_t)$. The policy parameters can be learned by directly maximizing the soft value function in Equation (17).

$$J_\pi(\theta) = \mathbb{E}_{a_t\sim\pi_\theta}[Q_\phi(s_t,a_t) - \alpha log\pi_\theta(a_t|s_t)]. \tag{20}$$

For the temperature factor α, it can be updated by minimizing the following loss function:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_\theta}[-\alpha log\pi_\theta(a_t|s_t) - \alpha\bar{\mathcal{H}}], \tag{21}$$

where $\bar{\mathcal{H}}$ is a desired minimum expected entropy, which can be designed as follows:

$$\bar{\mathcal{H}} = -\dim(\mathcal{A}). \tag{22}$$

Algorithm 1 summarizes the main process of SAC. There are two V-networks with the same architecture, one with the parameter φ inputs the state s_t and outputs the expected state value $V_\varphi(s_t)$, the other one with the parameter $\bar{\varphi}$ uses to evaluate the target state value $V_{\bar{\varphi}}(s_t)$ to compute the target Q value $\hat{Q}(s_t, a_t)$. Line 1 initializes the target V-network with the parameter of the expected

ALGORITHM 1: SAC Algorithm

INPUT: Initial policy parameters θ, Q-function parameters ϕ, V-function parameters φ, $\bar{\varphi}$, empty experience replay pool D with size M.

1: $\bar{\varphi} \leftarrow \varphi$.
2: **repeat**
3: Observe state s, select action $a \sim \pi_\theta(\cdot|s)$.
4: Execute a, get the transition $\langle s, a, r, s', d\rangle$.
5: Store (s, a, r, s', d) in D.
6: Reset environment state if d indicates terminal.
7: **if** it is time to update **then**
8: **for** each gradient step **do**
9: Randomly sample a batch of transitions $B = \{(s, a, r, s', d)\}$ from D.
10: Compute $J_V(\varphi)$, $J_Q(\phi)$, $J_\pi(\theta)$, $J(\alpha)$,
11: Update ϕ, φ, θ, α by back-propagation,
12: Update target network $\bar{\varphi}$.
13: **end for**
14: **end if**
15: **until** convergence

V-network. Lines 3 to 6 describe the process of interaction between agent and environment; the agent selects an action according to the policy after getting the environment state (line 3), then executes the action in the environment and gets a transition experience (line 4). The transition experience is stored in the experience pool (line 5), and the environment will be reset if it reaches a terminal state (line 6). When the experience pool accumulates a certain amount of samples, then we can use these samples to update the networks' parameters (lines 7–14). First, we randomly sample a batch of data from the experience pool (line 9), then we calculate the loss function of each network (line 10), the networks' parameters are finally updated by back-propagation (line 11). The target V-network is usually updated in the way of soft target update, that is, $\overline{\varphi} = tau \cdot \varphi + tau \cdot \overline{\varphi}$, where tau is a very small value, which means that we do not update the target network at once, but very little.

4.3 Summary of the Framework

Figure 3 shows the main framework of our model. At each time-step, the agent observes the current demand and supply condition and outputs the order price and drivers' schedule policy as its action. Then the number of induced customers and available drivers can be computed according to the action. In the sequential time-step, the agent will receive the reward from the environment and find itself in a new state. The agent repeatedly interacts with the environment, collecting a quantity of interaction experience, then the agent changes its policy to get a long-term reward.

We further give the overall computation details regarding how the SAC solves the dynamic pricing problem in Algorithm 2. The algorithm inputs the episode length T and the coefficient of contribution λ_1, λ_2, and outputs the optimal prices and schedule policy of each time-step. First, we initialize the vehicle's distribution at time-step $t = 0$, then the agent begins to interact with the environment during discrete time-step (lines 3–14). During the interaction process, the agent observes the environment and then executes the selected action (lines 5–6). A reward signal will be computed and fed back to the agent (lines 7–11), and the interaction data each time-step is put into the experience for subsequent network training (line 13). Lines 15–17 update network parameters. The algorithm repeats the above steps many times until convergence.

5 EXPERIMENTS

In this section, we conduct performance comparison experiments on real-world datasets. The proposed method is implemented using Python 3 and runs on a single machine equipped with Intel Core i7 processor quad-core and 24 GB DDR3-1333 DIMM RAM on Windows 10.

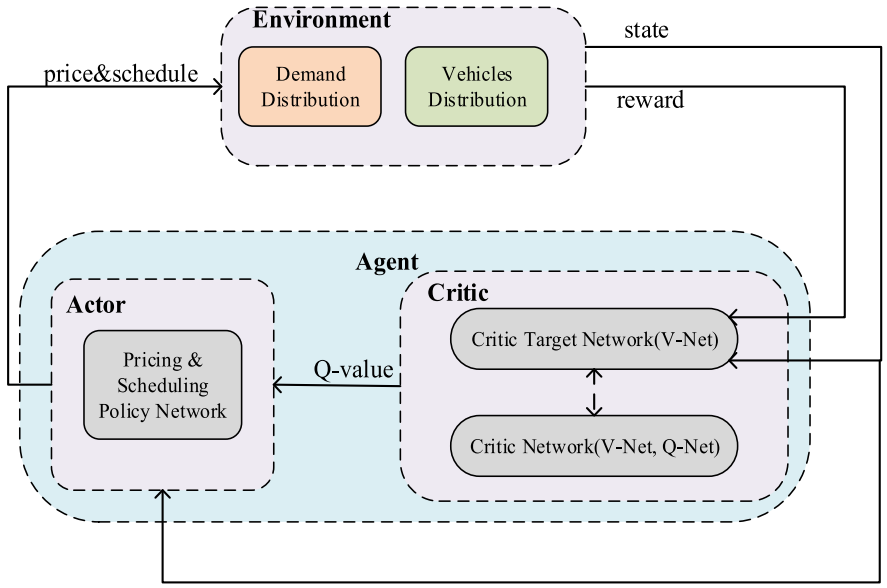


Fig. 3. Illustration of RL algorithm and pricing principles.

ALGORITHM 2: Dynamic Pricing Algorithm Based on SAC

INPUT: episode length T, Coefficient of contribution λ_1, λ_2.

Initialize the vehicles distribution at time-step $t = 0$.
repeat
  Reset environment state.
  for $t = 0 : T - 1$ **do**
    Observe $s_t = \langle C(t), S_v(t) \rangle$.
    Select a price and schedule policy $a_t = \langle P(t), B(t) \rangle$.
    Compute induced customers and drivers $D(t)$, $U(t)$ using Equations (1), (2).
    Compute the orders enabling to serve $O(t)$ using Equation (3).
    Compute $REV(t)$ using Equation (4).
    Compute $ORR(t)$, $SDR(t+1)$ using Equations (10), (11).
    $r_t = REV(t) + \lambda_1 ORR(t) + \lambda_2 SDR(t+1)$.
    $s_{t+1} = p(s_{t+1}|s_t, a_t)$.
    Store (s_t, a_t, r_t, s_{t+1}).
  end for
  if it is time to update **then**
    Update network parameters in SAC algorithm.
  end if
until convergence

OUTPUT: optimal prices and schedule policy $\langle P^*(t), \beta^*(t) \rangle$ from $t = 0$ to $t = T - 1$

5.1 Dataset

We conducted experiments based on real-world demand patterns (the yellow taxi data from the New York City Taxi and Limousine Commission dataset [3] for Monday, May 6, 2019), which are obtained from NYC.gov websites [2]. The dataset contains taxi trips generated per day, including fields capturing pick-up and drop-off dates/times, pick-up and drop-off region, and and so on. Case studies are conducted in Manhattan, which is divided into eight regions. Manhattan is one of the zones of New York City having dense travel trajectories, and more than 80% of travel trips take place in Manhattan.

5.2 Baselines and Experimental Setting

Baselines: To compare with existing algorithms and investigate the performance of our method in terms of total revenue and order response rate, we select and design several methods as baselines.

- **Current Origin Pricing Policy (COP):** An algorithm considers the current demand and supply in the network, optimizing price only by finding the maximum of $Rev(t)$. Compared with PODP, it does not consider the drivers' schedule and $Rev(t+1)$.
- DPEA: The algorithm [30] (Dynamic Pricing and Management for Electric Autonomous Mobility on Demand Systems Using Reinforcement Learning) is taken as one of the benchmark algorithms. This algorithm is aimed at the Dynamic pricing problem of electric vehicles to optimize the order price, vehicle scheduling ratio, and the strategy of empty car redirection. The reward function adopts instant reward, and the optimization algorithm adopts TRPO. After proper modification, the algorithm is applied to the dynamic pricing problem of online ride-hailing.
- **Predictive Origin-Destination Pricing Policy (PODP):** A dynamic pricing method that utilizes the networks' predicted future demand to further improve the generated revenue [5]. It optimizes the price and schedule jointly by solving a convex optimization function, which combines current revenue $Rev(t)$ with future estimate revenue $Rev(t+1)$.
- IMR: The vehicle scheduling ratio is not optimized. During the experiment, the distribution of vehicle scheduling ratio is consistent with the destination distribution of passenger demand. The reward function uses immediate income.
- REV: A reinforcement learning method that directly uses revenue at each time-step as the reward function. That is, we set $\lambda_1 = 0$ and $\lambda_2 = 0$.
- RORR: A reinforcement learning method that uses the revenue and order response rate as the reward function, which means that we set $\lambda_2 = 0$.
- RSDR: A reinforcement learning method that uses the current revenue and the KL divergence, which is the divergence from available drivers' distribution to potential customers' distribution at the next time-step, as the reward function. It means that we set $\lambda_1 = 0$.

Experimental Setting: For environment setting, one day is divided into 48 time slots, the distance between two time slots is 30 minutes, the episode length is $T = 47$. At $t = 0$, we initial the number of available drivers the same as the number of potential demand at time-step $t = 0$ and make its distribution following the uniform distribution. Both the COP and the PODP assume all trips starting at time t will end up at $t+1$ to facilitate the comparison with the baselines. We also assume all trips have the same travel time and set $p_{max} = 10.0$, $F^r(\cdot) = F^v(\cdot) = \frac{p^2}{p_{max}^2}$. For the contribution of KL terms, the default settings are $\lambda_1 = 0.2$ and $\lambda_2 = 1$, because the KL term of $ORR(t)$ is a big number, which will dominate the reward. While the term of $SDR(t+1)$ is relatively smaller than the term of revenue in reward function, thus, we keep its original value.

For the SAC algorithm, we use three neural networks with two hidden layers and 256 neurons in each hidden layer to approximate the Q-function, V-function, and policy function. The buffer size is 40,000. The delayed factors, learning rate, and batch size are set as 0.99, 0.0003, and 256, respectively. All methods based on reinforcement learning are trained for 1 million iterations using SAC algorithms, and parameters of SAC not specified are set by default. Specially, we use ROSDR to name the proposed method.

5.3 Results

5.3.1 Overall Performance Comparison. The pricing and scheduling strategies are evaluated five times in terms of APR and ORR, then the average values are taken as results. APR is the total

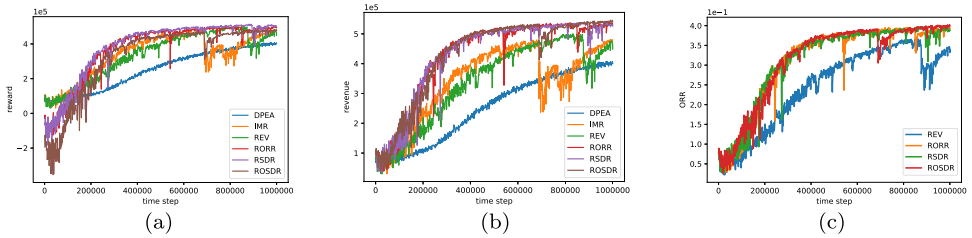
Table 1. Overall Performance Comparison in Terms of APR and ORR with Respect to COP

Metrics	APR	APR.Inc	ORR	ORR.inc
COP	453,609.6	-	0.30	-
DPEA	406,776.9	−10.3%	0.30	+0.0%
PODP	512,298.5	+12.9%	0.36	+20.0%
IMR	492,700.4	+8.6%	0.37	+23.3%
REV	475,343.9	+4.8%	0.35	+16.7%
RORR	548,480.3	+20.9%	0.41	+36.7%
RSDR	540,564.0	+19.2%	0.40	+33.3%
ROSDR	554,117.8	+22.2%	0.41	+36.7%

amount of revenue received at each time-step, and ORR is computed using the ratio between the number of served orders and the number of potential passengers. The APR increase and ORR increase with respect to the COP are also computed, and all results are showed in Table 1. It is obvious that methods considering the schedule of drivers and future demand have higher APR and ORR, and they achieve at least 4.8% and 16.7% improvement separately. However, the PODP only considers the demand at next time-step; it does not optimize the APR and ORR over the longer run. The several reinforcement learning methods, containing modified reward function and aiming to maximize the total amount of reward received over one day, achieve a significant increase in terms of both APR and ORR. The APR and ORR of ROSDR increase by 22.2% and 36.7%, much higher than the rest methods based on RL. Among the methods based on RL, the performance of REV is worse than the PODP, validating that it is not appropriate to directly use revenue as reward function for the dynamic pricing problem. The increase of RORR and RSDR is less than ROSDR, which indicates that the common contribution of the $ORR(t)$ and $SDR(t+1)$ can further help the agent find better policy.

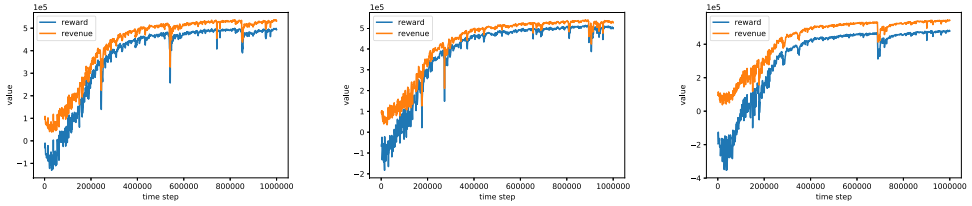
For the DPEA method, due to the dynamic pricing of electric autonomous mobility being different from taxi assignment, the results are weak. For technical reasons of the weak results, there are two reasons shown as follows: The first one is that TRPO is used in DPEA. TRPO is on-policy learning that the collected samples are used only once to update the policy, which faces poor sample efficiency. DDPG and SAC in our article are off-policy learning that the collected samples can be reused many times to update the policy. However, the DDPG will struggle to obtain good results in high-dimensional tasks, and it suffers from brittleness due to the temperature hyperparameter of action noise, while SAC has three key advantages compared to the other two DRL methods. First, it is an off-policy framework that reuses experience for data efficiency. Second, it maximizes the expected reward function and entropy to enable stability and exploration. Third, it can automatically tune the hyperparameter to avoid the trouble of manually tuning parameters. The second reason is that this baseline uses only revenue as a reward. So, this baseline is weaker than our proposed method.

5.3.2 Learning Process Analysis. The learning curves of different methods are shown in Figure 4(a). Because these methods adopt different reward functions, it does not make sense to compare the value of episode rewards, while the trend of curves makes sense. The REV that directly uses revenue as reward function converges slower than other methods that use modified reward function. The methods with modified reward function converge faster and are more stabilized. We also visualize the episode revenue and episode ORR generated during the training process, which are shown in Figures 4(b) and 4(c). It is obvious that methods with modified



(a) (b) (c)

Fig. 4. The episode reward (a), episode revenue (b), and episode ORR (c) of different methods during training phases.



(a) RORR (b) RSDR (c) ROSDR

Fig. 5. The changes of episode revenue relative reward in different methods.

reward functions have high episode revenue and episode ORR. In the scenario of ride-hailing platform, price affects the behaviors of passengers and drivers. High price can encourage more drivers to participate in the system while it can harm the willingness to pay for passengers. The large gap between demand and supply will eventually decrease the revenue. The fine tuning of prices or schedules may have little impact on revenues, which do not let the agent know the right optimization direction very well, so the agent needs to make a thorough exploration. When $ORR(t)$ and $SDR(t+1)$ are introduced into the reward function, those values start with a big and negative number, thus making the reward also start with a large negative number. This gives the agent a strong signal and guides the agent to counterweight between passengers and drivers.

Figure 5 shows the changes of episode revenue relative reward in different methods. For all methods with the modified reward functions, the trend of two curves keeps the same basically, which shows that the reward function newly designed can correctly guide the agent to optimize its goals. The gap between reward and revenue is large at first, because $ORR(t)$ or $SDR(t)$ is a big, negative value, dominating the reward. To improve the reward, the agent needs to learn how to balance the demand and supply as far as possible. As observed in the figures, the agent gradually learns to narrow the gap.

5.3.3 Parameter Sensitivity Analysis. Our method has two key hyper-parameters, i.e., (i) the parameter λ_1 of $ORR(t)$ and (ii) the parameter λ_2 of $SDR(t+1)$. In the experiment, the term of $ORR(t)$ is often a multiple of revenue, dominating the reward and greatly influencing the performance. While the term of $SDR(t+1)$ is much smaller than revenue. Therefore, we fix the $\lambda_2 = 1.0$ and investigate how the proposed method performs with the change of λ_1, studying the sensitivity of λ_1.

Figure 6 illustrates the sensitivity of λ_1, which depicts the increase of APR and ORR for ROSDR with respect to PODP at different λ_1. We observe that different λ_1 will result in different performance, and no regular pattern is observed. However, higher ORR usually corresponds to higher APR, which is necessary to introduce $ORR(t)$ into the reward function. On the whole, the

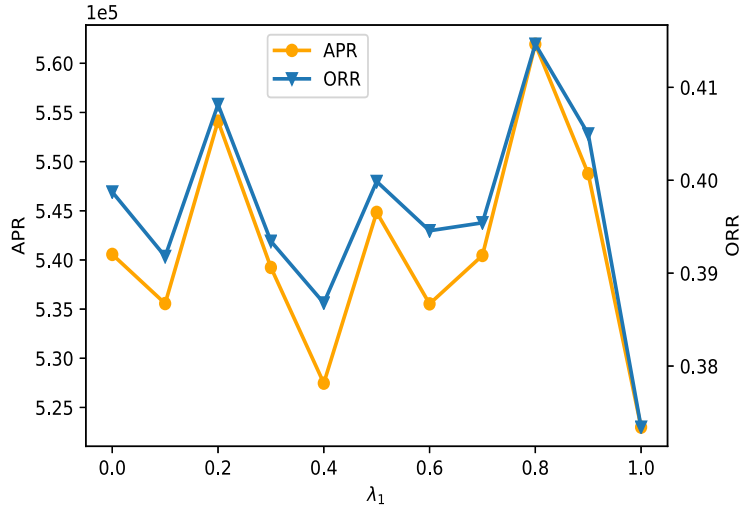


Fig. 6. The performance of ROSDR in terms of APR and ORR under different λ_1 settings.

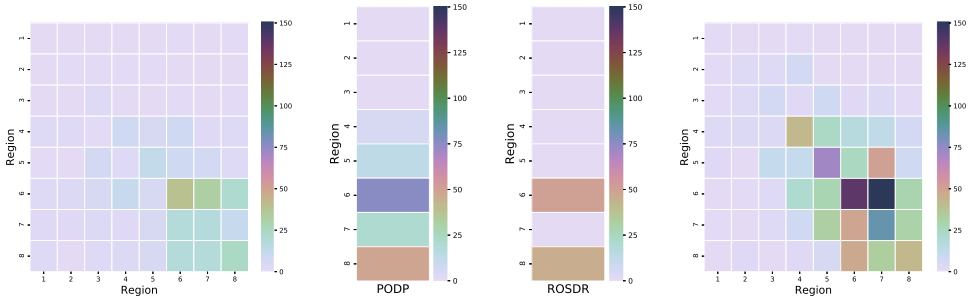


Fig. 7. (a) The potential demand matrix at 10:00AM. (b) The difference vector at 10:00AM between potential demand and idle vehicles in each region, following the policy of PODP. (c) The difference vector at 10:00AM between potential demand and idle vehicles in each region, following the policy of ROSDR. (d) The potential demand matrix at 10:30AM.

performance is influenced by the parameter of λ_1, thus the ride-hailing platforms should carefully select the hyper-parameters to generate higher performance.

5.3.4 Policy Analysis. To better understand the differences between the proposed method and baselines, we further analyze the policies of different methods. We conduct case study and take PODP as baseline. The state includes potential demand and vehicles' status, in which potential demand at each time-step is same for ROSDR and PODP, while vehicles' status is determined by the scheduling policy of each method. Figure 7(a) depicts the potential demand matrix at 10:00AM; it is obvious that most potential demand distributes among regions 6, 7, and 8. We use the difference vector to reflect the demand-supply balance condition; each element of vector is the difference between the number of potential demands and the number of idle vehicles of each region. Figures 7(b) and 7(c), respectively, reflect the demand-supply balance condition following the policies of PODP and ROSDR, in which the darker the color is, the bigger the gap between demand and supply. The color of ROSDR is lighter than of PODP, which indicates that the demand-supply condition resulted from the policy of ROSDR is superior to PODP.

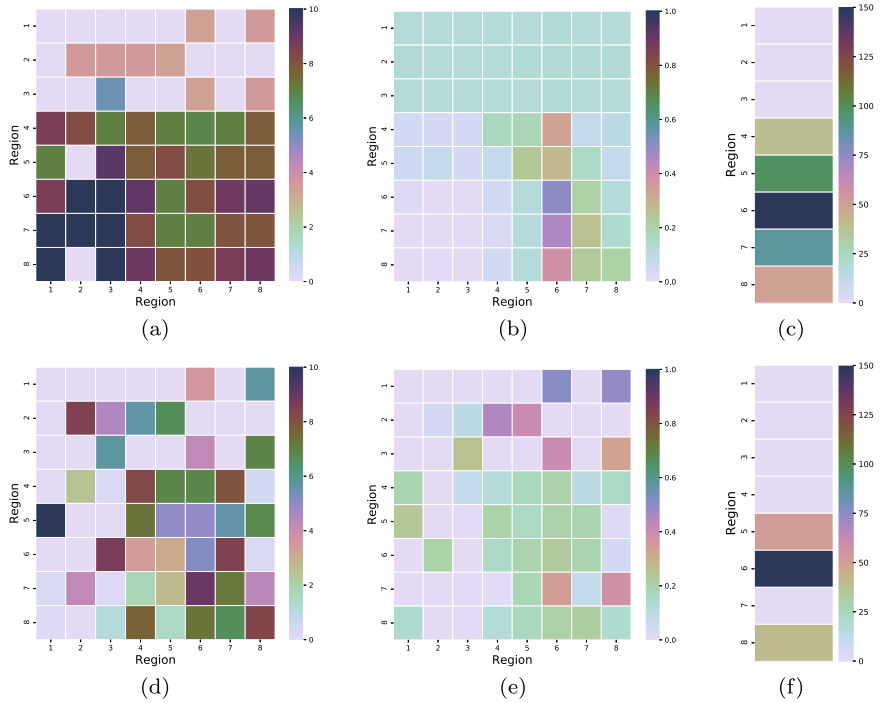


Fig. 8. (a) and (b) are pricing policy and scheduling policy of PODP, (c) is the difference vector at 10:30AM between potential demand and idle vehicles in each region obtained by following the policies of PODP. Similarly, from (d) to (f) are pricing policy, scheduling policy, and different vector of ROSDR.

Figure 8 illustrates the policies generated by PODP and ROSDR in their respective state at time-step 10:00 AM. Figures 8(a) and 8(d) are the respective pricing policies; it is obvious that the overall color of PODP is darker than the color of ROSDR, which means that the prices charged by PODP are higher than ROSDR. For PDOP, regions 6, 7, and 8 are in shortage of supply at time-step 10:00 AM, and the demand at next time-step is rising, which can be observed from Figure 7(d). PODP takes the demand-supply at the next time-step into consideration, it will incentivize more trips to those regions with high demand by setting relatively low prices for orders with a destination at those regions and scheduling more idle drivers to serve these orders. This idea can be validated from the price matrix and schedule matrix of PODP. PODP sets many high prices for orders with a destination in regions 1, 2, and 3, trying to suppress the demand to those regions and schedules few idle vehicles to serve these orders, further reducing the movement of the vehicle to these three regions. However, from Figure 8(c), the demand-supply balance condition at the next time-step is still poor. For ROSDR, there are only two regions in shortage of supply at time-step 10:00AM, thus ROSDR has no need to set a high price to suppress demand or reduce the schedule of vehicles to low demand region. From Figure 8(f), the demand-supply condition resulted from ROSDR is obviously superior to PODP. Therefore, the price followed by the policy of ROSDR is relatively lower than PODP, but also the demand-supply condition is better than PODP, which validates the superiority of the proposed method.

5.3.5 Efficiency. When the number of regions is small, our algorithm can compute the dynamic price efficiently. As the number of regions become greater, both the state dimension and action

dimension increase, which requires bigger networks and more training time, resulting in weak efficiency.

However, our algorithm can be transformed into a multi-agent based model conveniently, each region in the traffic network uses an agent to optimize the order price and vehicle scheduling ratio from this region to other regions, and all agents work together to achieve the global goal. Because agents are homogeneous, we can apply the parameter-sharing technique and further train the SAC model to all agents simultaneously. In this way, the number of networks to be trained not only does not increase, but also the dimension of action is reduced. The multi-agent-based model for dynamic pricing is one of our future works.

6 CONCLUSION

In this article, we propose a reinforcement learning method for dynamic pricing of the ride-hailing platforms. We define the pricing process as a Markov Decision Process, giving the definitions of state, action, and reward functions. This article is first to employ deep reinforcement learning for dynamic pricing problem of the ride-hailing platform. Compared with the mathematical optimization-based methods that focus on finding the maximum of short-term objective, the proposed method considers maximizing long-term goals when making decisions (pricing and scheduling). In practice, the movement of drivers affects future spatial distribution of available drivers, thus it is more forward-looking by optimizing long-term objective. Moreover, instead of using instantaneous revenue as the reward function, the ORR that represents the current service quality and the KL divergence that reflects the future demand-supply condition are introduced into the reward function, guiding the agent to find correct direction of optimization faster. Specifically, to match the scale of ORR and revenue, we redefine the ORR by the KL divergence from the served order distribution to the available driver distribution. Finally, we conduct experiments to evaluate the performance of the proposed method on the metrics of APR and ORR. Results have demonstrated that our method achieves higher APR and ORR than the baselines. Besides, results show that directly using revenue as reward function in dynamic pricing problem will not work well, while the newly proposed reward function can give the agent a positive signal. So, it can learn how to optimize price and drivers' schedule and improve service quality. Due to the new reward function, the policy converges faster and better, resulting in a significant increase of APR and ORR.

REFERENCES

[1] Didi Chuxing. Retrieved from http://www.didichuxing.com/en/.
[2] New York City Taxi and Limousine Commission Dataset. Retrieved from https://www1.nyc.gov/site/.
[3] New York City Trip Data. Retrieved from https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
[4] Uber. Retrieved from https://www.uber.com.
[5] Mohammad Asghari and Cyrus Shahabi. 2018. ADAPT-pricing: A dynamic and predictive technique for pricing to maximize revenue in ridesharing platforms. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 189–198.
[6] Siddhartha Banerjee, Ramesh Johari, and Carlos Riquelme. 2015. Pricing in ride-sharing platforms: A queueing-theoretic approach. In *Proceedings of the 16th ACM Conference on Economics and Computation*. ACM, 639–639.
[7] Matthew Battifarano and Zhen Sean Qian. 2019. Predicting real-time surge pricing of ride-sourcing companies. *Transport. Res. Part C: Emerg. Technol.* 107 (2019), 444–462.
[8] Kostas Bimpikis, Ozan Candogan, and Daniela Saban. 2019. Spatial pricing in ride-sharing networks. *Oper. Res.* 67, 3 (2019), 744–769.
[9] Juan Camilo Castillo, Dan Knoepfle, and Glen Weyl. 2017. Surge pricing solves the wild goose chase. In *Proceedings of the ACM Conference on Economics and Computation*. 241–242.
[10] M. Keith Chen and Michael Sheldon. 2016. Dynamic pricing in a labor market: Surge pricing and flexible work on the Uber platform. In *Proceedings of the 2016 ACM Conference on Economics and Computation*. 455.
[11] Yiwei Chen and Ming Hu. 2018. Pricing and matching with forward-looking buyers and sellers. *Rotman School Manag. Work. Pap.* 2859864 (2018).

[12] Harish Guda and Upender Subramanian. 2019. Your Uber is arriving: Managing on-demand workers through surge pricing, forecast communication, and worker incentives. *Manag. Sci.* 65, 5 (2019),1995–2014.
[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).
[14] Byung-Gook Kim, Yu Zhang, Mihaela Van Der Schaar, and Jang-Won Lee. 2015. Dynamic pricing and energy consumption scheduling with reinforcement learning. *IEEE Trans. Smart Grid* 7, 5 (2015), 2187–2198.
[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
[16] Jiaxi Liu, Yidong Zhang, Xiaoqing Wang, Yuming Deng, and Xingyu Wu. 2019. Dynamic pricing on e-commerce platform with deep reinforcement learning. arXiv:1912.02572 [cs.LG]
[17] Renzhi Lu, Seung Ho Hong, and Xiongfeng Zhang. 2018. A dynamic pricing demand response algorithm for smart grid: Reinforcement learning approach. *Appl. Energ.* 220 (2018), 220–230.
[18] Roberto Maestre, Juan Duque, Alberto Rubio, and Juan Arévalo. 2018. Reinforcement learning for fair dynamic pricing. In *Proceedings of SAI Intelligent Systems Conference*. Springer, 120–135.
[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[20] Markus Peters, Wolfgang Ketter, Maytal Saar-Tsechansky, and John Collins. 2013. A reinforcement learning approach to autonomous decision-making in smart electricity markets. *Mach. Learn.* 92, 1 (2013), 5–39.
[21] Xinwu Qian and Satish Ukkusuri. 2017. Time-of-day pricing in taxi markets. *IEEE Trans. Intell. Transport. Syst.* (01 2017). DOI: https://doi.org/10.1109/TITS.2016.2614621
[22] Rupal Rana and Fernando S. Oliveira. 2014. Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning. *Omega* 47 (2014), 116–126.
[23] J. Rempel. 2014. A review of Uber, the growing alternative to traditional taxi service. *AFB AccessWorld® Mag.* 51, 6 (2014).
[24] Gavin Adrian Rummery. 1995. *Problem Solving with Reinforcement Learning*. Ph.D. Dissertation. University of Cambridge.
[25] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Netw.* 61 (2015), 85–117.
[26] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning*. 1889–1897.
[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[28] Weiran Shen, Binghui Peng, Hanpeng Liu, Michael Zhang, Ruohan Qian, Yan Hong, Zhi Guo, Zongyao Ding, Pengjun Lu, and Pingzhong Tang. 2020. Reinforcement mechanism design: With applications to dynamic pricing in sponsored search auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 2236–2243.
[29] Richard S. Sutton, Andrew G. Barto et al. 1998. *Introduction to Reinforcement Learning*. Vol. 135. The MIT Press, Cambridge, MA.
[30] Berkay Turan, Ramtin Pedarsani, and Mahnoosh Alizadeh. 2019. Dynamic pricing and management for electric autonomous mobility on demand systems using reinforcement learning. *arXiv preprint arXiv:1909.06962* (2019).
[31] Shuoyao Wang, Suzhi Bi, and Ying Jun Angela Zhang. 2019. Reinforcement learning for real-time pricing and scheduling control in EV charging stations. *IEEE Trans. Industr. Inform.* 30, 4 (2019), 2149–2159.
[32] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. PhD Thesis. University of Cambridge, England.
[33] Chiwei Yan, Helin Zhu, Nikita Korolko, and Dawn Woodard. 2019. Dynamic pricing and matching in ride-hailing platforms. *Nav. Res. Logist.* 67, 8 (2019), 705–724.
[34] Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. 2019. 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*. 2645–2653.

Received January 2021; revised April 2021; accepted July 2021