Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

AN INTELLIGENT FRAMEWORK TO MAXIMIZE INDIVIDUAL TAXI DRIVER INCOME

Fang Chen
Hua Cai

School of Industrial Engineering
Purdue University
315 N Grant Street
West Lafayette, IN 47906, USA

Hong Wan

Department of Industrial and Systems Engineering
North Carolina State University
915 Partners Way
Raleigh, NC 27607, USA

ABSTRACT

The ridesharing platform has significantly changed how taxis operate in recent years. Most previous works focus on improving the user experience and maximizing the revenue from the platform or system level. The individual driver benefits are rarely addressed. In this work, we propose a deep reinforcement learning-based framework to help the individual driver maximize their daily income via order selections and self-repositioning. We first formulated the taxi operation as a Markov Decision Process. Then we created a multi-agent simulation consisting of the taxi drivers that use different strategies. A deep Q network-based (DQN) framework is proposed for drivers to learn which orders to select and where to reposition. Our result shows the driver who adopts DQN framework outperforms all other drivers. Furthermore, we found that the optimal policy does not suggest the driver operating in particular areas but recommends selecting the order with $5 to $7.5 taxi fare.

1 INTRODUCTION

Taxis play an essential role in the urban transportation system. Traditionally, vacant taxis cruise around the city and get hailed directly by passengers on the street. By meeting and picking up passengers on the street, taxi drivers can only rely on their driving experience accumulated over the years and some luck to find passengers to maximize their daily incomes. The fuel and time costs when taxis are vacant and the lack of experience in seeking passengers on the streets could significantly lower the number of orders being served and drivers' daily incomes.

Thanks to smartphones being widely used over the last decade, ridesharing platforms, such as Uber, Lyft, DIDI Chuxing, etc., are able to track both drivers' and passengers' locations through smartphones' GPS and then match drivers with passengers using its optimized algorithm (Smith 2016). More specifically, unlike traditional taxi-hailing services, ridesharing platforms do not allow passengers to hail their taxis directly on the streets. Alternatively, passengers send their trip request information, including pick-up locations, drop-off locations, and pick-up times, to the ridesharing platform through ridesharing platform websites or Apps. Based on the trip request information received from passengers combined with available drivers' geographical information, passengers are matched with available drivers through the platform's optimal matching algorithm, and trip requests are directly dispatched to corresponding drivers. Besides matching passengers with the closest drivers, the ridesharing platforms ask vacant taxi drivers to reposition/relocate their taxis to higher-demand areas for serving passengers. Compared to the traditional taxi-hailing operation mode, even though taxi drivers lose some freedom in terms of selecting which orders to serve (order selection) or deciding which areas to go (self-reposition), the ridesharing platform still has many advantages of reducing passengers' waiting time and drivers' idle cruising time.

As the ridesharing platform provides a better user experience for passengers, improving taxi revenue has become a key issue. Rong et al. (2017) developed a strategy by mining the historical trip records and exploring important factors in drivers' incomes. The overall income can be increased by predicting future taxi rides. Pandit et al. (2019) used a queuing model to compare static and dynamic pricing strategies for ridesharing services in a small region and concluded dynamic pricing strategy is able to achieve higher overall revenue. Wu et al. (2021) proposed a queuing model to simulate drivers and a dynamic pricing model to maximize the platform profit. Asghari and Shahabi (2018) developed a dynamic pricing mechanism by considering the system's future demand to increase the platform's overall revenue. These studies concentrate on mining historical trip data. The input of trip data is not mapping directly with a decision the driver makes at each time. By developing the Markov Decision Process (MDP) model and using reinforcement learning (RL) based approaches (Sutton and Barto 2018), drivers can learn the optimal decisions by interacting with a dynamic environment using the trial-and-error method. For instance, vacant taxis can be learned to reposition to better locations. Zhou et al. (2018) proposed a novel Spatial Network-based Markov Decision Process (SN-MDP) model solved by dynamic programming to recommend better driving directions for vacant taxis. Chen et al. (2020) proposed an online taxiing framework to maximize profits using a new fuel consumption estimation model. Jiao et al. (2021) developed a deep reinforcement learning-based intelligent driver assistant program for idling drivers to reposition themselves to achieve better income efficiency, i.e., income-per-hour. Instead of considering drivers as agents, Liu et al. (2021) considered each grid cell as an agent and proposed a multi-agent reinforcement learning-based framework, META (MakE Taxi Act differently), to mitigate the disequilibrium of supply and demand via taxi repositioning. The city-level revenue is then improved since more orders are served after repositioning. To maximize daily income, vacant taxi drivers are not only able to reposition themselves but also make decisions on whether to serve passengers or not. Gao et al. (2018) applied a Q learning-based solution to help taxi drivers to make operation choices among empty driving, carrying passengers, or waiting. Zhou et al. (2018) proposed an RL approach to help drivers choose between serving and idling. Most previous studies focus on maximizing all taxis' system-level or global revenue using reinforcement learning-based or other methods. However, only a few studies target individual driver income maximization, and the one we found is from Wang et al. (2020), where a deep Q network based-approach was proposed to recommend orders to drivers. Specifically, they created a model to predict and evaluate the Q-values of nearby orders using a supervised learning scheme based on historical trip data. No actual agent-based simulation was built for agents to learn the optimal solution through interacting with the environment. So the influence of other agents' decisions was not evaluated.

Before 2022, to attract more passengers to use their platforms and ridesharing services, ridesharing platforms tend to focus more on improving passengers' user experience, i.e., reducing passenger waiting time, lowering the trip request rejection rate, etc. As a result, drivers are not allowed to select which order they want to serve, and they cannot even see orders' details, e.g., drop-off locations, estimated trip time, and fares, before accepting passengers' orders. Since the middle of the last year, the ridesharing platform, such as Uber, enhances drivers' user experience by letting them choose the trip they want (Capoot 2022). Specifically, drivers are granted permission to view all nearby orders and then are allowed to select which order they want to take according to their personal preferences. Since there are thousands of trip orders daily, it is hard for the individual driver to decide which order is optimal at the current time and location. Maximizing daily income becomes a sequential decision-making problem. To help drivers select the 'best' orders among many orders at each time, we create a fully distributed multi-agent simulation containing various types of drivers and propose a Deep Q Network (DQN) decision-making algorithm to help individual drivers maximize their daily income through order selection and self-repositioning.

The rest of the paper is organized as follows. In Section 2, we introduce our distributed agent-based simulation model as well as our proposed DQN-based algorithm. In Section 3, the result and our model's performance will be discussed. Our work is concluded in Section 4 with a discussion of the limitations of our work and potential research directions that may arise.

2 METHOD AND SIMULATION MODEL

This section discusses trip data preprocessing, the agent-based simulation model assumptions and settings, and the proposed deep Q network (DQN) based framework for the individual taxi driver to maximize their daily income. In general, taxi drivers served as agents in the simulation. We first develop a multi-agent Markov Decision Process (MDP) simulation model consisting of only rule-based agents that adopt various order selection policies. We run scenarios for the multi-agent simulation model to record rule-based agent trajectories. Then the agent who adopts the reinforcement learning algorithm to learn the optimal policy, known as the RL agent, is trained using the rule-based agent trajectories recorded in the pre-run scenarios. More details are included in the following subsections.

2.1 Trip Data Preprocessing and Key Assumpionts

The trip data we used is historical yellow cab taxi orders obtained from New York City Taxi & Limousine Commission website (NYC TLC). The dataset is one of the most common public data sets in the transportation research area. The data includes pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, and fares. Pick-up and drop-off locations are taxi zones pre-defined by the NYC government based on zip codes. Instead of applying input modeling to generate taxi orders, our simulation model directly uses historical taxi orders as real trip requests for agents to select. After picking an order, the agent transit from the current location to other locations and receive trip fares. We used the first two weeks of August 2017 data for training the model and the third week of August 2017 for testing. Based on the data we have, some key assumptions for the simulation model are listed below:

- **Time Step**: Previous studies regarding the dispatch optimization in the ridesharing system using reinforcement learning often round time-related features, i.e., pick-up time, drop-off time, and trip time, to the nearest 15, 20, or 30 minutes. However, we found many trips in the historical data lasting less than 15 minutes. We also consider the 15-minute waiting time might be too long for the vacant taxi that does not take an order in the previous time step. As a result, we have decided to round the pick-up time, drop-off time, and trip time to the nearest 5 minutes. The simulation time step, denoted by $t \in \{0, 1, 2, ..., 143\}$, starts with 0, corresponding to 7 a.m., and ends with 143, corresponding to 7 p.m. Since passengers are assumed to wait up to 15 minutes, orders at 6:45 a.m., 6:50 a.m., and 6:55 a.m. could also be retrieved and selected by agents.
- **Trip Time Estimation**: The trip time regarding a vacant taxi relocating from the current zone to the other zone is estimated using the average trip time between two zones based on historical trip data. If a vacant taxi serves an order, the trip time is the exact time shown on the order. In addition, a trip time of fewer than 5 minutes is considered 5 minutes to avoid taxis trapping in the same time step forever.
- **Passenger Waiting Time**: We assume passengers are willing to wait up to 15 minutes. In other words, a taxi driver can see and select any unserved orders within the past 15 minutes. If an unserved order showing in the order book has exceeded 15 minutes, it will be automatically dropped from the order book.
- **Taxi Operating Hours**: In New York City, a single taxi driver often has a 12-hour shift each day (Jula 2016). Based on this information, we assume all taxis to be operated 12 hours from 7:00 a.m. to 7:00 p.m., Monday to Sunday.
- **Taxi Operating Areas**: Only the Manhattan area's orders are considered, i.e., all taxi trips are limited within the Manhattan area. There are 69 taxi zones in total in the Manhattan area as of 2017 based on the NYC TLC website.
- **Removed Outliers from Dataset**: The orders whose trip time is longer than 90 minutes or shorter than 1 minute are considered outliers. Orders that are more than 200 USD are removed as well.

2.2 Problem Formulation as a Markov Decision Process

We model the taxi driver operation process, including order selections and passengers picking up and dropping off, as a Markov Decision Process (MDP) (Puterman 1990). The MDP is defined as (S, A, P, R, γ), where S is a finite set of states, A is a finite set of actions, P is a state probability transition matrix, R is a reward function, and $\gamma \in (0,1]$ is a discounted factor used in the reward function.

All taxi drivers are considered agents in the system. The simulation model has two categories of agents: rule-based agents and reinforcement learning (RL) agents. In general, rule-based agents serve as background agents, following a pre-designed policy to take actions and transit from the current state to the next. On the other hand, an RL agent does not follow the pre-designed policy but learns the optimal policy that maps states to actions based on the current observed environment to maximize its cumulative rewards by the end of the day. More details regarding the simulation setting, how agents interact with the environment, and how RL agents learn the optimal policy are discussed as follows.

2.2.1 Agent

Our simulation has two types of agents: rule-based agents and reinforcement learning (RL) agents. Specifically, rule-based agents make decisions following pre-designed rules, while RL agents learn their optimal policy through trial-error methods.

All agents have two basic attributes, a unique *id*, and a current *status*. $agent.id \in \{0,1,...,N\}$ where N is the number of agents, and is the identity of each agent. The $agent.status \in \{0,1\}$ is a binary variable that represents whether an agent can take an action. The $agent.status = 0$ means an agent is either self-repositioning or serving an order at the current time step. When $agent.status = 1$, each agent can select an order or reposition itself to a particular taxi zone. Besides basic attributes, the agent's state space is the key attribute of the agent, which will be described in the following subsection.

2.2.2 State

The state space for an RL agent is the observed environment at the current time step t. We refer to the state space in previous ridesharing studies and design our state space to include the current agent's spatial-temporal features and aggregated features from orders which can assist drivers in making decisions. The RL agent state space at the current time step t contains two types of information, agent basic spatial-temporal features, and unserved order aggregated features. The spatial-temporal feature contains the taxi zone where the agent is currently located and the current time step when $agent.status = 1$. Since we assumed that passengers could wait for up to 15 minutes, i.e., three-time steps, an available agent can select one unserved order placed within the past 15 minutes at the current taxi zone. The order aggregated features are the demand and average taxi fares from the current taxi zone to others. They are aggregated based on unserved orders placed within the past 15 minutes. Specifically, the demand feature is aggregated by counting the orders from the current taxi zone z_i to each taxi zone j. The average taxi fare feature is aggregated by calculating the average fares among orders from the current taxi zone z_i to each taxi zone j. As a result, the state is defined as

$$S_t = (X_t, D_{ij,t}, F_{ij,t})$$

- $X_t = (z_i, t_d, day)$, where $z_i, i \in \{0,1,2,...,68\}$ is the taxi zone number that the RL agent currently located when $agent.status = 1$; t_d is either the drop-off time step when the RL agent finishes serving an order or the arrival time step when the RL agent performs self-repositioning; the *agent.status* will be changed from 0 to 1 when $t_d = t$. *day* is the day of the week.
- $D_{ij,t}, j \in \{0,1,2,...,68\}$ is a 1×69 matrix. It represents the total number of available orders from the current zone z_i to each zone j within the past 15 minutes.
- $F_{ij,t}, j \in \{0,1,2,...,68\}$ is a 1×69 matrix. It represents the average fares $\bar{f_{ij}}$ of available orders from the current zone z_i to each zone j within in the past 15 minutes.

To increase the speed of the simulation and save CPU memory, the state space of rule-based agents only includes the X_t. Specifically, the state space of a rule-based agent is defined as $S_t = (z_i, t_d, day)$. We reduce agent state space because rule-based agents mainly serve as background agents, and we only need to track their locations and which orders they select at each time step.

2.2.3 Action

At each time step t, all available agents, i.e., $agent.status = 1$, take actions sequentially. Each available agent is limited to seeing and selecting the order whose pick-up zone is the same as the agent's current zone because traveling from the current zone to other zones often takes more than 15 minutes. The action space for each agent is defined as $A = \{a_j\}$, where $j \in \{0, 1, ..., 68\}$. More specifically, since Manhattan has 69 different zones, there are 69 drop-off zones. We categorize orders into 69 types according to the drop-off taxi zones. Each type of order refers to an action. If an action corresponds to multiple orders in the current state, the agent will select the order with the highest taxi fare. All agents are allowed to choose any unserved orders placed within the past 15 minutes sequentially.

Rule-based agents take actions following pre-designed rules. Three different types of rule-based agents are proposed. A "random agent refers to an agent who randomly selects an available order. A "max agent" adopts a greedy strategy. They always select the available order with the highest taxi fare. A "restricted agent" refers to an agent limited to picking up, dropping off passengers and relocating itself within the three busiest taxi zones. The three busiest taxi zones, also known as restricted taxi zones for restricted agents, are identified as having the highest trip requests among the three zones. Compared to random and max agents, the restricted agent only operates in areas with the highest demands. They aim to reduce the trip time per order and plan to serve as many orders as possible to improve their daily income.

2.2.4 Reward

Each agent receives an immediate reward after picking up an order. The immediate reward is the taxi fare associated with the order. If agents decide to relocate to other zones, the reward is zero. Then the immediate reward for an agent moving from zone i to zone j at timestep t is defined as $r_t = f_{ij} I_t$, where $I_t \in \{0, 1\}$ is an indicator to determine whether the agent is serving an order or repositioning. Specifically, $I_t = 0$ refers to the agent repositioning to other zones despite available orders; $I_t = 1$ refers to the agent accepting an order and receiving the taxi fare as a reward. The future discounted reward is defined as $R_t = \sum_{t=t'}^{T} \gamma^{t'-t} r_t$, where $T = 143$ is the terminated state and $\gamma \in (0, 1]$ is the discounting factor. We set the discounting factor as 0.99, allowing the agent to consider long-term rewards while not completely ignoring immediate ones.

2.2.5 Deep Q learning

Given the large state and action space, it is impossible to calculate the probability transition matrix P. A model-free Q learning algorithm (Watkins and Dayan 1992) is used to help the agent learn the optimal policy. By applying the neural network (Mnih et al. 2013) to estimate Q values, a model-free deep Q network (DQN) based algorithm is proposed for the RL agent to learn the optimal policy. More specifically, the DQN utilizes a deep neural network to approximate the Q function, denoted by $Q(s_t, a_t)$. The Q function, known as the state-action value function, estimates the expected cumulative reward for taking action a in a state s. The optimal Q value following any policy π, denoted by $Q^*(s_t, a_t)$, is the maximum Q value. It is defined as follows:

$$Q^*(s_t, a_t) = max_\pi E[R_t | s_t = s, a_t = a, \pi]$$

where $\pi(a|s) = P(A_t = a | S_t = s)$ is a probability distribution mapping from states to actions. By using the Bellman optimality equation, the Q-value can be updated iteratively as follows:

$$Q^{new}(s_t, a_t) = (1 - \alpha) Q^{old}(s_t, a_t) + \alpha (R_t + \gamma max_a(s_{t+1}, a))$$

where α is the learning rate.

The $Q^*(s_t, a_t)$ can be estimated using the deep neural network with weight θ, denoted by $Q(s_t, a_t; \theta) \approx Q^*(s_t, a_t)$. The loss function of the Q-network is the mean squared error of predicted Q-values and target Q-values. We use the loss function, denoted by $L_k(\theta_k)$, to update weight θ at each iteration k,

$$L_k(\theta_k) = E[((R_t + \gamma max_a Q(s, a; \theta_k^-)) - Q(s, a; \theta_k))^2]$$

where θ^- is the weight of the target Q-network. Notice that the weight of the target Q-network is not learned. Instead, θ^- is synchronized from θ of the Q-network in some previous iterations.

2.3 Proposed Algorithms

Algorithm 1: Rule-based Agents Simulator

```
Input: Agents, OrderBook;
Output: Locs, Rws;
for episode=(1, 15) do
    Set day = episode;
    Filter order based on day from the OrderBook;
    Initialize each agent states S_0 in Agents;
    Initialize Locs, Rws;
    Shuffle the order selection/decision-making sequence for Agents;
    for t = (0, 143) do
        for agent in Agents do
            if t ≠ t_d then
                continue
            end
            if agent.type = random then
                Select a random order if there are available orders;
                Select a random zone to reposition if there is no available order;
                Record the drop-off zone or reposition zone as a_t;
            end
            else if agent.type = max then
                Select the order that has the max fare if there are available orders;
                Select a random zone to reposition if there is no available order;
                Record the drop-off zone or reposition zone as a_t;
            end
            else
                Select the order within the three restricted zones if there are available orders;
                Select a random zone within three restricted zones to reposition if there is no
                  available order;
                Record the drop-off zone or reposition zone as a_t;
            end
            Update S_t based on a_t;
            Update Locs, Rws
        end
    end
end
```

DQN model is hard to converge, especially when the state-action pair space is large. Since our model consists of a large number of states and actions, based on previous experience, more than 10000 episodes are expected to use for training. In addition, given a large number of agents need to be iterated and updated at each time step, it takes a significant amount of time to finish running one episode. As a result, it is nearly impossible to conduct the full training with all 8000 agents together. We separate the RL agent training process and the rule-based agents running process to solve the long training time issue. In general, we first run scenarios for rule-based agents to track and store their trajectories. Then we incorporate and input rule-based agent trajectories in the RL agent model. Specifically, we run three different scenarios/replications with different random numbers for 8000 rule-based agents on each date from August 1st to August 15th. For each scenario, a matrix, denoted by $Locs \in Z^{T \times N}$, where $N = 8000, T = 143$, stores all rule-based agent trajectories at each time step. A $Rws \in R^{H \times N}$ matrix where H is the number of episodes is also used to store agents' cumulative rewards at the end of each episode. More details about the rule-based simulation model are demonstrated in the following paragraph.

Algorithm 2: RL Agent DQN based Order Selection Algorithm

Initialize replay memory D, Q-network parameter θ, Q-target parameter θ^-, *Agent*;
for *episode=1:H* **do**
  Select a random *day* from 1 to 15;
  Filter the *Orderbook* based on *day*;
  Select a random scenario *sce* for *day*;
  Filter rule-based agents trajectories *Locs* based on *day* and *sce*;
  Shuffle the order selection sequence for *Agents*;
  for *t=0:143* **do**
    if $t=t_d$ **then**
      Obtain available orders based on *Locs*;
      Aggregate $D_{ij,t}, F_{ij,t}$;
      Create a feature vector $\phi_t = \phi(s_t) = \phi(X_t, D_{ij,t}, F_{ij,t})$;
      Compute $Q(\phi_t, a')$ for all $a' \in A_t$;
      With probability ε select a random action a_t;
      With probability $1-\varepsilon$ select $a_t = max_a Q(\phi_t, a)$;
      Update ϕ_{t+1} based on action a_t;
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D;
      Sample random minibatch of transitions $(\phi_t, a_t, r_t, \phi_{t+1})$ from D;
      Minimize $E[((R_t + \gamma max_a Q(s_{t+1}, a_t; \theta^-)) - Q(s, a_t; \theta))^2]$ w.r.t θ using gradient descent method;
      Set $\theta = \theta^-$ every n step;
    end
  end
  $\varepsilon = \varepsilon \times decay\_rate$;
end

At the beginning of the episode, we first filter trip orders from the order book based on the selected day. Then all agents are located in random zones, and *Locs* are initialized based on that. A random decision-making sequence is set for agents to take action sequentially. At each time step, agents follow the random sequence to take actions and only the agent whose status is one can take action. Based on the type of agents, agents take actions differently. A random agent selects a random order with available orders or repositions to a random zone if there is no available order. A max agent selects an order with the maximum fare if there are available orders or goes to a random zone if there is no available order. A restricted agent

can only select an order in his three restricted zones if there are available orders or repositions to one of the three restricted zones if there is no available order. An agent updates its state, rewards, and status based on the action taken, and the $Locs, Rws$ are also updated accordingly. The complete algorithm for running rule-based agents and getting their trajectories is shown in Algorithm 1.

After obtaining the trajectories of rule-based agents in $Locs$, we train an RL agent by incorporating $Locs$ with the simulator. Specifically, for each episode, we randomly select a scenario of rule-based trajectories on a random day and assume that all rule-based agents follow the same trajectories stored in $Locs$. At the timestep that the RL agent can take action, we first obtain unserved orders based on other agent trajectories $Locs$. According to the random decision-making sequence, orders selected by the rule-based agents that have a higher sequence are dropped from the $OrderBook$. The RL agent is only able to observe and select the rest of the available orders. The order selected by the RL agent will be duplicated if selected by a lower sequence rule-based agent in $Locs$. Since there is only one RL agent in our model and there are more than 100000 orders daily, duplicating an order cannot significantly affect our simulation. Next, we aggregate demand and average fare by each zone for unserved orders and formulate the state space ϕ_t. ϕ_t served as the input of a two-layer fully connected neural network, and the output is $Q(\phi_t, a)$ values for all actions. With the probability $epsilon$, a random action is chosen. Otherwise, the action that has the largest Q value is selected. Based on the selected action, we update ϕ_{t+1} and store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in a replay buffer. To update θ and θ^- of Q-network and target Q-network, we randomly sample a minibatch of transitions from D, and minimize the loss function respective to θ using the gradient descent method (Kingma and Ba 2014). The target network weight θ^- is not learned through the neural network but is synchronized from the Q-network every N learning step. The ε starts with 1, then decreases by a decay rate until the minimum value of 0.05 is achieved. The complete algorithm is shown in Algorithm 2.

3 EXPERIMENT AND RESULT

3.1 Experiment

The distribution of temporal-spatial data usually changes over time. Agents observe different training and testing environments if the training and testing data set has different patterns or trends. Then RL model performance on the testing data set can be significantly affected. To better split the training and testing data sets and find which time period throughout the year can be used for training and testing, we train a simple binary classifier and use the AUC-ROC score to evaluate. More specifically, the AUC-ROC score is often used to evaluate how well the classifier can distinguish between two classes. We label our training data set 0 and testing data set 1. If the AUC-ROC score is close to 0.5, we conclude that our classifier cannot identify the training and testing data sets, which means they have similar patterns. By further analyzing the 2017 trip data using the method we discussed above, we have found adjacent weeks tend to have more similar patterns. We have decided to use the first three weeks of the trip data in August. The first two weeks and the third week have a 0.512 AUC-ROC score, which implies they have similar patterns and could be used as training and testing datasets.

We utilize the first two weeks of August 2017 trip data to train our RL agent. One day is considered one episode. At the beginning of each episode, a random day is drafted from the first two weeks of August. RL agent is trained under 13000 episodes containing over 25000 learning steps. The learning steps are not equivalent to the simulation time steps. It counts the number of time steps that the RL agent takes an action. There are 8000 rule-based agents and one RL agent in our simulation model. We assume that half of the rule-based agents are random agents, ten are restricted agents, and the rest are maxed agents. We use Python to construct our simulation model and use agent trajectories to validate the correctness of the MDP process. As mentioned above, we run three scenarios of the rule-based agent for each day separately to record their trajectories using three different random numbers. The randomness will first affect the agent's initial taxi zones, decision-making sequences, and random actions, which results in agents' trajectories and served orders changing. Then our RL agent is trained based on the rule-based agent trajectories. The RL

agent model is trained three times with three different random numbers to tackle the randomness issue. Three optimal policies will be evaluated in our test data set. It takes about 6 hours to finish running one scenario for rule-based agents and 10 hours to train the single-agent RL model. The total time for running all scenarios decreases when the parallel computing mechanism is applied to run different scenarios on multiple cores separately.

3.2 Result

Our result concludes that both average cumulative maximum Q values and average cumulative daily scores are able to converge, and those values from the RL model with the random number 41 are presented in Figure 1 below. According to the ZipRecruiter yellow cab salary in New York City dataset (ZipRecruiter 2023), a New York City yellow cab taxi driver can make an average of 25 USD/hour. This does not include leasing fees, insurance, fuel cost, etc., but only considers incomes collected from taxi fares. The top 2% drivers can make from 36.53 USD/Hour to 39.13 USD/Hour. In our model, the RL agent gets more than 40 USD/ Hour in training, which can rank in the top 2% in practice.

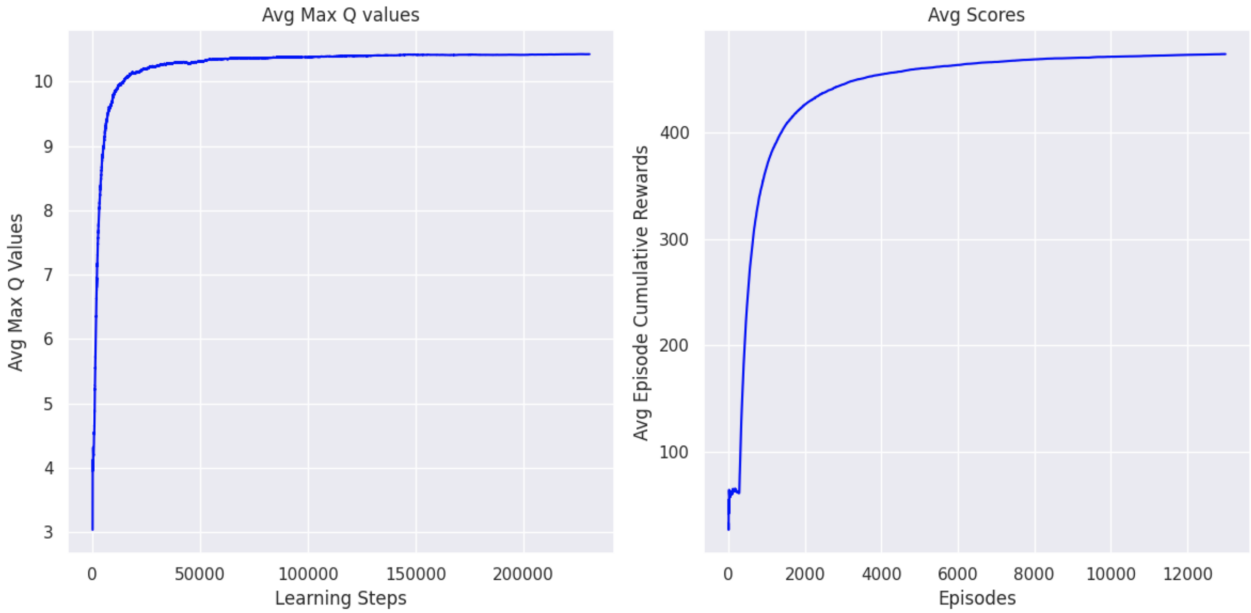

Figure 1: Training average maximum Q values.

The third week of August 2017 trip data is used for testing. Each day is considered one episode, and each episode is run with six replications, which is enough to construct the confidence intervals. We first compare our RL agent with three different types of rule-based agents, and the result is shown in Figure 2. Our RL agent significantly outperforms all rule-based agents based on the average daily scores of each agent from Sunday to Monday. Considering the randomness of the simulation model and the number of rule-based agents, extreme values can easily affect the average scores of each type of rule-based agent. We also calculate the percentile of scores for each type of agent. Table 1 compares three types of rule-based agents and the RL agent according to the percentiles. Our RL agent still significantly outperforms rule-based agents in all five different percentiles. Averaging across all days, the RL agent is able to make 43 USD/Hour, which is similar to our training result. Figure 3 presents the confidence intervals of the RL agent scores for each day of the week, Sunday through Monday, using different random numbers. Even though the average daily scores on various days of the week given the same random number and the average daily scores on the same day of the week among different random numbers are very similar, the ranges of confidence intervals

Table 1: Agent performance comparison.

Percentile	Random Agent	Max Agent	Restricted Agent	RL Agent
0	20.10	13.55	230.21	387.50
25	113.46	116.74	323.49	486.20
50	55.66	157.12	345.94	506.05
75	217.01	220.05	391.25	538.13
100	401.37	422.40	441.85	644.36

are different. Based on the first plot in Figure 3, A smaller confidence interval range, e.g., Wednesday to Friday, indicates a lower uncertainty. In comparison, a much larger confidence interval range, e.g., Sunday to Tuesday, results in much higher uncertainty. We conclude that our RL model is more reliable for the RL agent from Wednesday to Friday than the rest of the days.

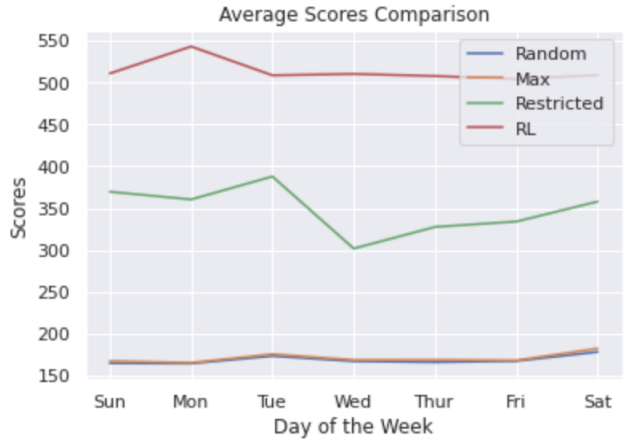

Figure 2: Average scores comparisons.

By further analyzing the record of the RL agent trajectories and selected orders, we want to explore what kind of orders the RL agent prefers to serve and what taxi zones the RL agent wants to operate. By testing three trained RL models on the testing data set, we found that the optimal policy obtained by the RL agent might not be the most related to the particular taxi zones where the RL agent picks up orders. Instead, it is more highly correlated to the taxi fare of each order. Figure 4 presents the taxi fare distributions among three random numbers. In conclusion, the optimal policy reflects that the RL agent tends to select orders with \$5 to \$7.5 and rarely selects orders with over \$15., leading to higher daily cumulative rewards. Short trip orders appear to be more attractive to the RL agent than long trip orders. Besides the fact that more short trip orders exist in the historical trip record, we think the intuition behind selecting \$5 to \$7.5 orders could be the driver can serve more orders within the time frame and earn more taxi fare initial charges, which make up a significant portion of the total fare. Another intuition could be longer trips increase the risk of traffic congestion. The earning per minute significantly reduces if drivers encounter traffic congestion.

4 CONCLUSION AND FUTURE DIRECTION

We developed a fully distributed agent-based simulation model for New York City taxi drivers and proposed a deep Q network-based approach to help individual drivers maximize their daily incomes through order selection. Compared with previous studies focusing on improving the passenger user's experience or maximizing the overall ridesharing platform revenue, our work aims to solve the individual driver income maximization problem using deep reinforcement learning. We compare the RL agent performance with

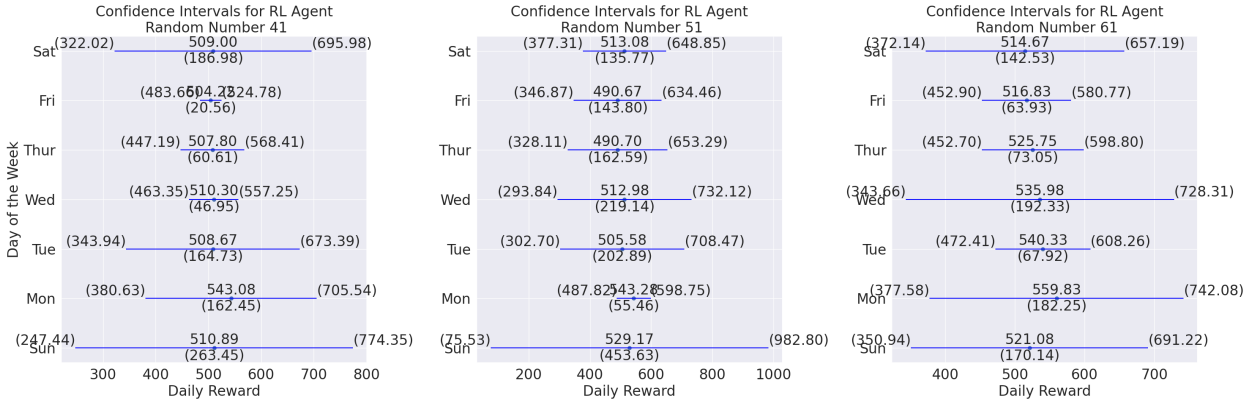

Figure 3: Daily rewards using three different random numbers.

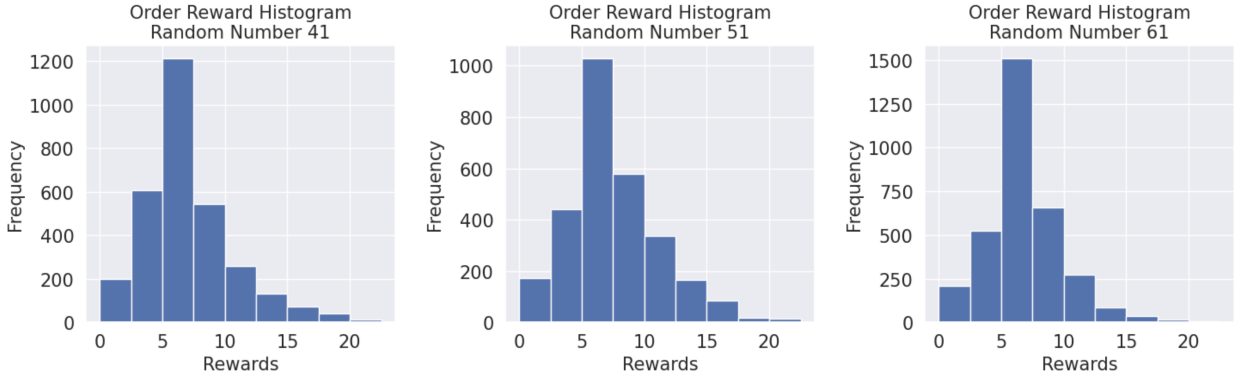

Figure 4: Daily rewards using three different random numbers.

8000 rule-based agents. The results show that the RL agent outperforms rule-based agents by adopting the optimal order selection policy.

Our simulation model and the proposed method still have several limitations. First, given that the research focuses on the driver experience aspect, we assume that if an unserved order has passed 15 minutes, the system will lose this order/customer. As a result, the rider experience is not addressed. We plan to study rider experience and driver preference tradeoffs in our future research, and also explore how much money needs to be added to those lost orders so that they can be served eventually during the 15-minute period. Secondly, since the current fully distributed model takes much time, we run three rule-based agent scenarios on each day separately and utilize agents' trajectories to train the RL agent. The RL agent only learns from three scenarios for each day, but there are much more scenarios in practice. As a result, the environment for training the RL agent is less dynamic than the real-world environment. Additionally, our current research creates a DRL-based framework for single agent. The obtained optimal strategy might not work when there are many RL agents. We plan to develop a multi-agent DRL-based model and obtain a joint optimal policy for more drivers to use in the future.

REFERENCES

Asghari, M., and C. 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, November 6th-9th, Seattle, USA*, 189–198.

Capoot, Ashley 2022. "Uber Unveils New Features, Including One that Lets Drivers Choose the Trips They Want". Available at https://www.cnbc.com/2022/07/29/uber-will-let-drivers-choose-the-trips-they-want-to-take.html, Accessed 29th July 2022.

Chen, H., B. Guo, Z. Yu, A. Wang, and C. Zheng. 2020. "The Framework of Increasing Drivers' Income on the Online Taxi Platforms". *IEEE Transactions on Network Science and Engineering* 7(4):2182–2191.

Gao, Y., D. Jiang, and Y. Xu. 2018. "Optimize Taxi Driving Strategies Based on Reinforcement Learning". *International Journal of Geographical Information Science* 32(8):1677–1696.

Jiao, Y., X. Tang, Z. T. Qin, S. Li, F. Zhang, H. Zhu, and J. Ye. 2021. "Real-world Ride-Hailing Vehicle Repositioning Using Deep Reinforcement Learning". *Transportation Research Part C: Emerging Technologies* 130:103289.

Jula, Megan 2016. "New York Taxi Drivers Denounce Proposed Shift Limits". Available at https://www.nytimes.com/2016/06/24/nyregion/new-york-taxi-drivers-denounce-proposed-restrictions.html, Accessed 24th June 2016.

Kingma, D. P., and J. Ba. 2014. "Adam: A Method for Stochastic Optimization". *arXiv preprint arXiv:1412.6980*. Available at https://arxiv.org/abs/1412.6980, Accessed 30th January 2017.

Liu, C., C.-X. Chen, and C. Chen. 2021. "Meta: A City-Wide Taxi Repositioning Framework Based on Multi-Agent Reinforcement Learning". *IEEE Transactions on Intelligent Transportation Systems* 23(8):13890–13895.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning". *arXiv preprint arXiv:1312.5602*. Available at https://arxiv.org/pdf/1312.5602.pdf, Accessed 19th December 2013.

Pandit, V. N., D. Mandar, M. K. Hanawal, and S. Moharir. 2019. "Pricing in Ride Sharing Platforms: Static Vs Dynamic Strategies". In *2019 11th International Conference on Communication Systems & Networks (COMSNETS), 7th-11st, January, Bengaluru, India*, 208–215. IEEE.

Puterman, M. L. 1990. "Markov Decision Processes". *Handbooks in Operations Research and Management Science* 2:331–434.

Rong, H., Z. Wang, H. Zheng, C. Hu, L. Peng, Z. Ai, and A. K. Sangaiah. 2017. "Mining Efficient Taxi Operation Strategies from Large Scale Geo-Location Data". *IEEE Access* 5:25623–25634.

Smith, A. 2016. "Shared, Collaborative and On-Demand: The New Digital Economy". Available at https://www.pewresearch.org/internet/2016/05/19/the-new-digital-economy/, Accessed 19th May 2016.

Sutton, R. S., and A. G. Barto. 2018. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, Cambridge, MA.

Wang, H., H. Rong, Q. Zhang, D. Liu, C. Hu, and Y. Hu. 2020. "Good or Mediocre? A Deep Reinforcement Learning Approach for Taxi Revenue Efficiency Optimization". *IEEE Transactions on Network Science and Engineering* 7(4):3018–3027.

Watkins, C. J., and P. Dayan. 1992. "Q-learning". *Machine learning* 8:279–292.

Wu, Y., W. Li, Q. Yu, and J. Chen. 2021. "Modeling the Impacts of Driver Income Distributions on Online Ride-Hailing Services". *Mathematical Problems in Engineering* 2021:1–9.

Zhou, X., H. Rong, C. Yang, Q. Zhang, A. V. Khezerlou, H. Zheng, Z. Shafiq, and A. X. Liu. 2018. "Optimizing Taxi Driver Profit Efficiency: A Spatial Network-based Markov Decision Process Approach". *IEEE Transactions on Big Data* 6(1):145–158.

ZipRecruiter 2023. "Yellow Cab Salary in New York City, NY". Available at https://www.ziprecruiter.com/Salaries/Yellow-Cab-Salary-in-New-York-City,NY#Hourly, Accessed 2nd September 2023.

AUTHOR BIOGRAPHIES

FANG CHEN is a Ph.D. student in the School of Industrial Engineering at Purdue University. His research interests include machine learning and reinforcement learning applications for the smart transportation system and agent-based simulation. His email address is chen1427@purdue.edu.

HUA CAI is an Associate Professor at Purdue University, with a joint appointment in Industrial Engineering and Environmental and Ecological Engineering. Her research interests include data-driven system modeling and optimization, prospective environmental assessment of emerging technologies, complex adaptive systems, and decision-making for sustainable consumption. Her email address is huacai@purdue.edu, and her website is https://engineering.purdue.edu/uSMART.

HONG WAN is an Associate Professor in the Department of Industrial and Systems Engineering at North Carolina State University. Her research focuses on the areas of experimental design and data analysis of complex simulation models and blockchain. She is the director of the ISE blockchain lab and serves as the editor-in-chief of the Journal of Blockchain Research and the associate editor of ACM TOMACS. Her email address is hwan4@ncsu.edu and her website is https://www.ise.ncsu.edu/people/hwan4/.