Seeking in Ride-on-demand Service: A Reinforcement Learning Model with Dynamic Price Prediction

Suiming Guo[a], Chao Chen[b]

[a]*Jinan University, 510632, Guangzhou, Guangdong, China*
[b]*Chongqing University, 400044, Chongqing, China*

Abstract

Recent years witness the development and increasing popularity of ride-on-demand (RoD) services such as Uber and Didi. Compared with traditional taxi service, RoD service is "more data-driven" and adopts dynamic pricing mechanism, of various forms, to manipulate the supply and demand in real time. The dynamic price could be viewed as an accurate and quantitative indicator of the supply and demand, and could provide clues to drivers, passengers, and the service providers, possibly reshaping the ways in which some problems are solved. In this paper, we focus on the seeking route recommendation problem that aims at recommending highly profitable or efficient seeking routes to drivers of vacant cars, and incorporate the effects of dynamic prices. We first justify our motivation by showing the importance of route recommendation and answering why it is necessary to consider dynamic prices that rapidly fluctuate throughout a day, based on the analysis of real service data. We then design a dynamic price prediction model to generate the dynamic prices at any given time and location based on features extracted from multi-source urban data. After that, a reinforcement learning model is adopted to perform seeking route recommendation based on real service data. We conduct extensive experiments in different spatio-temporal combinations and make comparisons with different baselines. Results not only validate that our dynamic price prediction model is highly accurate, but also prove that taking the real-time predicted dynamic prices into consideration significantly increases both driver revenue and utilization rate than merely using the average dynamic prices or completely ignoring dynamic prices.

Keywords: Ride-on-demand, dynamic price, driver revenue, reinforcement learning,

1. Introduction

During the last decade, ride-on-demand (RoD) service such as Uber and Didi first emerged as a new and disruptive mode of transport compared with traditional taxi service, and has gained increasing popularity in the market since then. RoD service is either convenient or beneficial for passengers and drivers. For passengers, it is more convenient to request for service through smart phones, and the price is more affordable; for drivers, in many cities it is not required to apply for licenses or medallions before driving, and working schedules are more flexible.

RoD service has two unique features, namely, the use of *dynamic pricing*, and being *data-driven*.

Dynamic pricing. Dynamic pricing, also known as "surge pricing" when it was introduced in Uber, aims to manipulate the supply and demand in real time, i.e., the service provider sets a higher price when demand – potential passenger requests – exceeds supply – vacant cars on the road, and vice versa. Possible forms of dynamic pricing include auction-based mechanisms, additive bonus to the trip fare, and multiplying the trip fare by a real-time factor (called "dynamic price multiplier"), etc. Most service providers choose the multiplicative form in practice, and so does our study.

Email addresses: guosuiming@email.jnu.edu.cn (Suiming Guo), cschaochen@cqu.edu.cn (Chao Chen)

Dynamic pricing could be viewed as a closed-loop feedback mechanism to manipulate the supply and demand: the price is determined based on the real-time supply and demand condition, and it, in turn, controls the supply and demand in a way leading to a higher service efficiency. Besides, dynamic prices could also be regarded as an accurate indicator of supply and demand, based on which drivers, passengers, and the service provider are now able to inspect changes on the road or design heuristics as well as algorithms to deal with such changes, e.g., improving the efficiency of seeking, scheduling or dispatching, providing certain recommendation service, etc.

Data-driven. The use of on-board GPS devices triggers a large amount of studies and applications in taxi service since the beginning of the last decade. Examples of studies based on GPS trajectories include inferring order origins and destinations, detecting possible detours, optimizing for driver revenue based on driving habits, etc.

RoD service goes one step further and tremendously expands the volume and variety of data. It is mainly based on smart phones and mobile apps, which is a data source providing much more abundant information. For example, accurate order information is now readily available, including origin, destination, the time getting on/off the car, the time and location the request is issued, text descriptions (typed by the user on the mobile app) of origin and destination, etc. User behavior log is also possible, as it could be recorded by the mobile app. Typical user behaviors include the type of mobile device, the number of times one checks for trip fare, the regions one zooms in and out to view available drivers, etc. All these new sources of data offer richer information than the traditional GPS trajectories, empowering researchers, service providers or any other third parties to design algorithms, conduct experiments and perform evaluations to make RoD service more efficient than before.

With these two new features, it is now possible to reshape the ways in which some traditional problems are solved. In this paper we focus on the seeking route recommendation problem, a widely studied problem in mobility service, especially taxi. Basically, seeking route recommendation aims at recommending profitable or efficient seeking routes to drivers of vacant cars. Drivers themselves may adopt some naive strategies based on experience or word-of-mouth. Such strategies may be inaccurate, and may even suggest a large number of drivers blindly flocking to the same hot spots such as the central business district, exacerbating the already imbalanced supply and demand condition. Algorithms developed in previous taxi-related studies solve this problem by finding local or global hot spots, modelling driver behavior using Markov decision process, simulating driver behavior using physics-based approaches, etc.

Things are different in RoD service. Firstly, the data-driven feature enables us to acquire more datasets, such as multi-source urban data, to solve the problem. Secondly, dynamic prices could be an accurate and readily available indicator of supply and demand, which previous studies go into great lengths to find out. Furthermore, to make full use of dynamic prices that fluctuate in real-time, it should be evaluated on a fine spatio-temporal granularity, e.g., using recent or average dynamic price multipliers may not be enough.

In this paper, we solve the seeking route recommendation problem by combining reinforcement learning with dynamic price prediction. We first give intuitive answers to two questions, i.e., *why recommending seeking routes* and *why considering dynamic prices*, based on the analysis of real service data. After that, a seeking route recommendation framework consisting of two parts is built. The first part is a dynamic price prediction model based on multi-source urban data, trying to obtain a dynamic price multiplier given the spatio-temporal condition and other relevant features when a driver picks up a passenger. Evaluation results show that our dynamic price prediction model achieves an accuracy ranging from about 83% to 90% in different settings, laying a solid foundation for the reinforcement learning model. In the second part, we use Markov decision process to model the seeking behavior, and adopt a reinforcement learning model to tackle the seeking route recommendation problem, in which the predicted dynamic prices from the first part are incorporated into reward design. We evaluate driver earnings and utilization rates under our model, and it proves that taking the real-time predicted dynamic prices into consideration significantly increases both metrics than merely using average dynamic prices or completely ignoring dynamic prices.

Our contributions are listed in the following:

- We summarize the two unique features of RoD service, i.e., dynamic pricing and data-driven, and solve a common problem, albeit widely studied in previous taxi service, by making full use of these

two features. Specifically, for the seeking route recommendation problem, we first build a dynamic price prediction model and then show that considering dynamic prices, especially the real time predicted dynamic prices, is beneficial for increasing driver earnings. By comparison, most previous works do not consider dynamic pricing and simply treat RoD service as a new version of taxi. Even the few works that indeed consider dynamic pricing only calculate the average or historical prices and ignore its real time nature.

- We adopt a reinforcement learning model to solve the seeking route recommendation problem, and incorporate dynamic prices into the model. The reinforcement learning model helps to consider the long term effects of seeking routes and thus redistributes drivers more effectively.
- We conduct extensive experiments and comprehensively evaluate both our dynamic price prediction and reinforcement learning model. Firstly, we use real service datasets from a typical RoD service in China, making our results and discussions more convincing and tenable. Secondly, for each model, we compare its performances with a number of state-of-the-arts and provide a detailed discussion.

The remainder of this paper is organized as follows. Section 2 reviews the related work. In section 3 we present our datasets and provide a detailed and intuitive analysis. Section 4 and section 5 discuss the dynamic price prediction and the reinforcement learning model, respectively. Section 6 presents a comprehensive evaluation on both models. Section 7 gives a brief summary and some discussions based on evaluation results. Finally, section 8 concludes the paper.

2. Related Work

We provide discussions on related work about three topics: RoD service, dynamic price and its prediction in RoD service, and seeking in taxi or RoD service.

RoD service. RoD service is also known as on-demand ride-hailing, and in some cases, people may call it as ride-sharing. In fact, "RoD" and "ride-sharing" have different emphases. Ride-sharing emphasizes the share of rides with the same or similar origins or destinations, either between passengers or between the driver and passengers. Such sharing was common during the early deployment of Uber, as it was an effective means to attract more drivers and increase market share. RoD, on the other hand, emphasizes "on-demand" – the service is available as soon as one asks for it at any time or location. Under this setting, RoD service is viewed as a disruptive new version of taxi, by supporting mobile app, adopting dynamic prices, and being data-driven. Our study focuses on RoD service, so we omit the discussions on ride-sharing.

Compared with taxi, RoD service is relatively new, and thus receives limited attention. Most studies simply treat RoD as a service similar to taxi, and want to find out the differences between them. For example, [1] focuses on the passenger waiting time and make a comparison, of both waiting time and price, between Uber and taxi; [2, 3] choose the market share as the study target; [4, 5] discuss the impacts and market effects of Uber's entrance – e.g., how driver behavior is changed since Uber takes place.

Dynamic price and its prediction in RoD service. Dynamic pricing plays an important role in many services, as an effort to either improve service efficiency or manipulate supply and demand in different forms. For examples, dynamic pricing is used in Internet retail [6], hotel pricing [7], flight ticket pricing [8, 9], inventory management [10], etc.

As one of the two unique features of RoD service, dynamic pricing and its effects are studied in a number of works. For example, [11, 12, 13] discuss the effects of dynamic pricing in balancing and redistributing supply and demand, increasing driver revenue, and reducing passenger waiting time. [14], as a typical early study of RoD or on-demand ride-hailing service, evaluates Uber's surge pricing as a black-box by placing simulated mobile app users across important locations. [15, 16] explore demand pattern, the effects of dynamic pricing on passengers, passenger behavior, and etc. [17] combines pricing with dispatching, which is a more traditional problem, and proposes a distributed pricing framework. Some studies take an economics perspective: examples that consider the effects of dynamic pricing on supply and demand include [11, 18, 19].

The problem of dynamic price prediction is also tackled using various methods. For example, [20] defines the predictability of price multipliers and uses Markov-chain or neural network models to predict average dynamic price multiplier of a region based on the predictability. [21, 22] turn to the prediction at a finer granularity and predict the dynamic price multiplier given time and location using linear regression or neural network based on multi-source urban data. [23] summarizes the above works, adopts an ensemble learning model, and chooses different models based on price multiplier predictability. [24], on the other hand, emphasizes the interpretability of price prediction results and presents a simple but quantifiable approach to dynamic price prediction. It gives a detailed evaluation as to what features have more obvious impacts on price multipliers.

Seeking in taxi or RoD service. Seeking analysis is also a highly visible topic in traditional taxi service, and in RoD service it also receives some, yet limited, attention. Generally speaking, we could divide such analysis into seeking strategies analysis and seeking route recommendation, the latter of which is the target of study in this paper.

Seeking strategies analysis could be considered as a macro-level problem, and tries to uncover the relationship between driver seeking strategies (e.g., choosing hot spots, driving faster, etc.) and revenue. For example, [25, 26] consider two different strategies, i.e., hunting or waiting for passengers, and compare their performances under different circumstances, based on taxi GPS trajectories. In RoD service, [27] collects multi-source urban data and designs a framework to mine the relationship between driver revenue and the carefully-crafted features that are relevant to seeking strategies. In such relationship, dynamic prices become an important component.

Seeking route recommendation could then be considered as a micro-level problem, and aims to recommend the right road segment or city cell a driver should keep seeking for so that driver revenue is increased. In these studies, Markov decision process is frequently used to model the interaction between drivers and the service itself – e.g., [28, 29, 30, 31]. Another example using Markov decision process is [32], and it mainly pays attention to electric taxis and make charging decisions based on both battery constraint and GPS trajectories. Besides Markov decision process, [33] generates recommendation results by minimizing the distance between taxis and potential passenger requests; [34, 35] apply reinforcement learning; [36] solves the problem by allowing a single driver to be matched to multiple passengers; [37] builds theoretical models and optimization problems. In RoD service, [38] uses Q-learning to recommend profitable seeking routes.

Different from the above works, our study combines the power of dynamic price prediction and seeking route recommendation. Firstly, we implement a more robust and accurate dynamic price prediction model and achieve a satisfactory prediction accuracy. More importantly, we explain why it is necessary to consider the predicted dynamic price multipliers in seeking route recommendation, and make a thorough evaluation about the effects of price prediction, of reinforcement learning, and of both, on driver revenue.

3. Data and Analysis

In this section, we first present the multi-source urban datasets used in our paper, and then provide a detailed data analysis to give intuitive answers that motivate our study.

3.1. Multi-source Urban Datasets

We categorize the multi-source urban datasets into RoD datasets, taxi dataset, and public datasets.

3.1.1. RoD Datasets

In a typical RoD service, both order request and creation are done on a mobile app, basically in the following procedure. The user first types the intended origins and destinations or chooses from the recommended locations, and then the mobile app sends back all the information and retrieves an estimated trip fare as well as the current dynamic price multiplier. The estimated trip fare is the product of dynamic price multiplier and a base fare that is determined by trip distance and duration. In our dataset, the dynamic price multiplier ranges from 1.0 to 1.6. The user then makes a decision, either accepting the estimated trip fare and creating an order, or giving up and possibly requesting again later.

Unlike previous studies on seeking route recommendation in taxi service that mainly rely on taxi GPS trajectories, in RoD service the "data-driven" feature makes it possible to obtain more datasets. In our study, we also acquire order data and dynamic price data besides GPS trajectories. These datasets are explained in details in the following.

GPS trajectories. This dataset is similar to the taxi GPS trajectories dataset that has been widely used before. It contains the periodic GPS records, in longitude and latitude, of every single car in operation. Each record includes the longitude and latitude, time stamp, speed, direction, the unique car ID, etc. This dataset spans from Aug. to Nov. 2016, and contains the records of about 3,500 to 3,800 cars daily in Beijing, China. Specifically, the ranges of longitude and latitude are $[116.21, 116.56]$ and $[39.81, 40.08]$.

Order and dynamic price data. The information of orders and dynamic prices are recorded in the same dataset. For the order data, in RoD service, the use of mobile app enables accurate recording of order information, as order origins and destinations are now clearly specified by users and recorded by the service provider. For the dynamic price data, the price multiplier is recorded when the service provider returns the estimated trip fare and current price multiplier. Sometimes getting the trip fare estimates does not mean order creation, and to accurately associate an order with a price multiplier, only the price multiplier returned in the trip fare estimate that is closest to order creation is retained. Our dataset also covers the time span from Aug. to Nov. 2016, and the total number of orders is $2,742,120$. Each entry includes origin, destination, the time getting on and off, the unique ID of passenger/driver/car/order, the estimated trip fare, price multiplier, etc.

In all the above datasets, all unique IDs of drivers, passengers and cars are anonymized so that one cannot relate an ID to a specific person or car.

3.1.2. Taxi Dataset

Besides RoD datasets, we also use taxi GPS trajectories as an auxiliary dataset. The reasons are straightforward. Firstly, as mentioned in [24], RoD and taxi service are similar and complementary to each other, and hence driver behavior should be similar. For example, a hot region in taxi service may also be good enough for RoD drivers. In other words, taxi GPS trajectories could serve as a useful guideline for RoD drivers. Secondly, taxi service data also describes car movements on the roads and is helpful in characterizing the traffic in different spatio-temporal combinations. For example, if the average speed of taxis is low, then it may be an indication of traffic congestion in a region. Our taxi GPS trajectories covers the same time span, and contains records of about 30,000 taxis in Beijing.

3.1.3. Public Datasets

As part of our multi-source urban datasets, we also acquire public datasets to provide more information in dynamic price prediction. There are a lot of possible choices, and in our study we choose the POI data and public transport distribution data, as described in the following.

POI data. POI information is widely used in studies on location-based service, and it characterizes the type of a location such as airport, restaurant, etc. Here we rely on POI data to describe the origins and destinations of orders. We crawl the POI data from AMap service, one of the most popular online map service providers in China. In our data, each POI falls in one of the 14 categories: car service, restaurant, shopping, sports & entertainment, hospital, hotel, scenic spot, business & residential building, government, education & culture, transportation facility, finance & insurance, business and lifestyle.

Public transport distribution data. Public transport services – e.g., bus, metro, tram, public rental bike – also play an important role in transportation, and the effects of considering public transport data are similar to that of considering taxi data. That is, the status of public transport services could give a hint on RoD driver behavior and describe traffic condition. In our study, we choose the distribution of bus and metro service as our public transport distribution data. Specifically, we acquire the locations of all bus (and metro) stations and all the buses or metros that stop by these stations, from AMap service, similar to our POI data. It should be noted that though the most accurate description of bus or metro distribution should be dynamic and recorded in real-time by, say, examining the smart-card usage data or the GPS trajectories of buses and metros, we consider it enough to simply count the stations and lines because such data are easily accessible and public transport services usually have fixed timetables.

3.2. Data Analysis

In data analysis, we give intuitive answers to two questions, i.e., *why recommending seeking routes* and *why considering dynamic prices*, for two purposes. Firstly, the data analysis helps to form a comprehensive picture and gain some understanding about RoD service. Secondly, it also provides inspiration for our study.

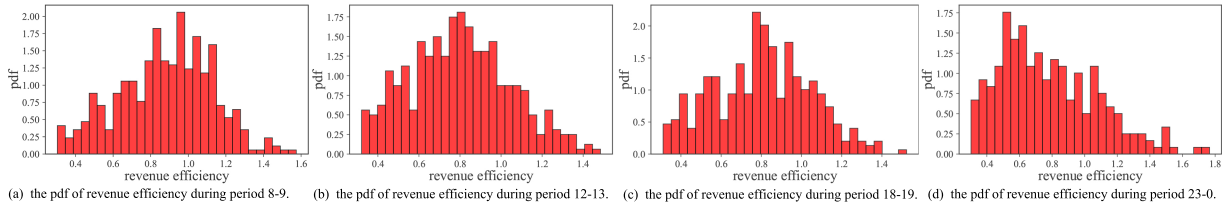
3.2.1. Why Recommending Seeking Routes?

Before presenting data analysis results, we first give the definition of driver revenue. The fare of a RoD trip, i.e., the driver revenue during this trip, is defined as,

$$r = dp_o \cdot (f_{base} + f_d * d_{trip}). \tag{1}$$

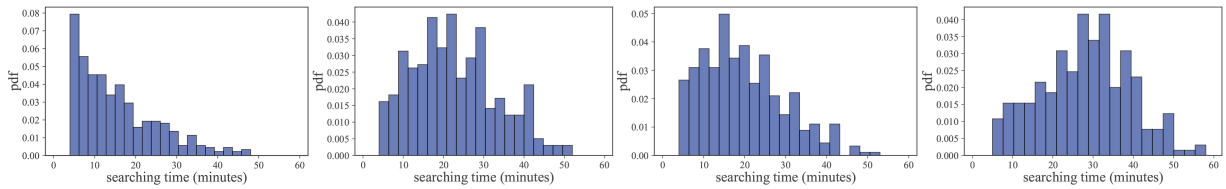
In (1), dp_o, f_{base} and f_d are the dynamic price multiplier at the trip origin, the flag-fall price, and the unit price per kilometre, respectively. d_{trip} is the trip distance from the origin to destination. This is an approximation, ignoring the extra fare of slow driving and minimum distance, and we consider it acceptable because these two terms are usually much smaller. In our dataset, we have $f_{base} = 15$ and $f_d = 2.8$, both in RMB Yuan. We also show the distribution of revenue efficiency during some specific time period, and it is simply the total revenue obtained during the period divided by the length of the period.

In our analysis, we first show the distribution of revenue efficiency among all drivers during four representative time periods, i.e., [8:00, 9:00), [12:00, 13:00), [18:00, 19:00), and [23:00, 0:00), in Fig. 1. Intuitively, these four time periods correspond to morning rush hour, non-rush hour around noon, evening rush hour, and late night hour. Similarly, in Fig. 2 we also show the distribution of searching time, i.e., the time a driver seeks for passengers between two consecutive trips, during these time periods. In addition, Tab. 1 summarizes some simple statistics – i.e., the 1st quartile, 3rd quartile, and average – of both revenue efficiency and searching time from Fig. 1 and Fig. 2.



(a) the pdf of revenue efficiency during period 8-9. (b) the pdf of revenue efficiency during period 12-13. (c) the pdf of revenue efficiency during period 18-19. (d) the pdf of revenue efficiency during period 23-0.

Figure 1: The probability distribution functions of revenue efficiency during four representative time periods.



(a) the pdf of searching time during period 8-9. (b) the pdf of searching time period 12-13. (c) the pdf of searching time period 18-19. (d) the pdf of searching time period 23-0.

Figure 2: The probability distribution functions of searching time during four representative time periods.

We have the following observations regarding Fig. 1, Fig. 2, and Tab. 1:

- Both revenue efficiency and searching time vary significantly across different time periods.
  - For example, in the morning rush hour, the average revenue efficiency and average searching time are the highest and the lowest, respectively, among all four time periods. The late night hour is just the opposite.

Table 1: The statistics of revenue efficiency and searching time during four representative time periods.

Time period	**Revenue efficiency** (Yuan/min)			**Searching time** (min)		
	1st quartile	3rd quartile	average	1st quartile	3rd quartile	average
[8:00, 9:00)	0.71	1.06	0.88	8	20	15.49
[12:00, 13:00)	0.62	0.96	0.80	15	29	22.63
[18:00, 19:00)	0.63	0.97	0.81	12	26	19.73
[23:00, 0:00)	0.54	0.98	0.78	20	36	28.14

  - Comparing between the late night hour and the evening rush hour (or the non-rush hour around noon), it should also be noted that the difference of searching time is much more obvious than that of revenue efficiency. This makes the seeking route recommendation problem more complicated.

- Both revenue efficiency and searching time vary significantly across drivers. For revenue efficiency, during [8:00, 9:00), the 3rd quartile is 49.30% higher (1.06 v.s. 0.71) than the 1st quartile, and during [23:00, 0:00), the percentage is much higher and achieves 81.48% (0.98 v.s. 0.54). For searching time, the comparison is similar.

- Therefore, it is necessary to recommend seeking routes to drivers. Firstly, a good seeking strategy or seeking route could help those lower-earning drivers become more efficient and earn more. Secondly, such recommendation should consider carefully as to what features influence the recommendation results. For example, our observations above shows that the temporal features are important enough.

3.2.2. Why Considering Dynamic Prices?

Previous studies on seeking route recommendation in taxi service usually go to great lengths in finding "hot spots" or finding locations with more high-earning orders. In RoD service, the dynamic price multiplier is an accurate indicator of the supply and demand condition – which is just the goal of using dynamic pricing – and it should be an integrated description of the information that previous studies want to find out.

To justify that it is necessary to consider dynamic prices in seeking route recommendation, in the following we show and explain the distribution of dynamic price multipliers on both temporal and spatial dimension. Fig. 3 show the temporal distribution of dynamic price multipliers on both weekdays and weekends. In Fig. 3, we calculate the average dynamic price multiplier among all orders every half an hour, in the whole city of Beijing.

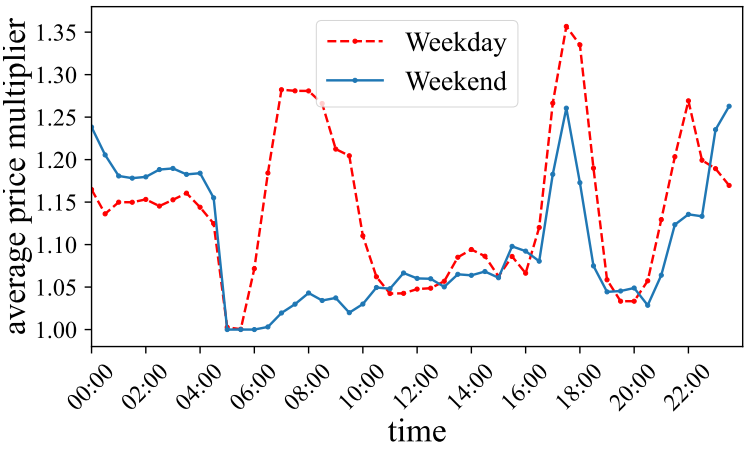


Figure 3: The temporal distribution of dynamic price multipliers.

It is clear from Fig. 3 that,

- The average dynamic price multiplier is greater than 1.0 in most of the time. In some time periods, e.g., the evening rush hours on weekdays, the average dynamic price multiplier reaches as high as 1.35. In other words, dynamic prices have a significant impact on driver revenue.
- The dynamic price multiplier has different patterns during different hours-of-day or days-of-week, and hence temporal features are important. For example, on weekdays, there are four obvious peaks: the small hours, the morning and evening rush hours, and the late night hours. On weekends, the patterns are different: there are not morning rush hours, and price multipliers remain at a low level until evening rush hours.
- The dynamic price multiplier fluctuates rapidly throughout a day. This is the result of its real-time nature, as it is designed to reflect the real time changes of supply and demand. As a result, when considering dynamic prices in our study, a fine temporal granularity is needed, and simply calculating the average or historical price multipliers is not enough.

On the spatial dimension, we focus on the evening rush hours [18:00, 19:00) on weekdays, divide the area of study (i.e., within longitude [116.21, 116.56] and latitude [39.81,40.08]) into 900 ($= 30 \times 30$) cells, and plot the spatial distribution of the number of orders (i.e., *demand*), dynamic price multipliers, and the number of vacant cars (i.e., *supply*) in Fig. 4, Fig. 5 and Fig. 6. It is shown that:

- The distributions of the number of orders as well as vacant cars are spatially imbalanced, and it is thus necessary to recommend seeking routes. These distributions agree with common intuitions: the number of orders and vacant cars are much higher in the city center, and are drastically reduced in city suburbs. One thing to notice is that the airport of Beijing is located at the upper right corner in these figures, and the vacant cars towards the upper right corner in Fig. 6 corresponds to those going to and from the airport.
- The distribution of dynamic price multipliers is also spatially imbalanced, but with more complicated patterns. Intuitively, the price multiplier is higher in the city center. Counter-intuitively, the price multiplier in the city suburbs surrounding the city center is even higher, indicating severe supply and demand imbalance.
- Comparing the above two observations, the reason of having higher price multipliers but fewer orders and cars in city suburbs is that drivers tend to flock to city center to seek for passengers, leaving a very limited number of drivers in suburbs. Though the orders are indeed fewer in suburbs, the number of drivers is still not enough to meet the demand. To the contrary, for those drivers going to city center, though there are indeed more orders, there are even more drivers, making price multipliers lower.
- Therefore, if dynamic price multipliers are observed and considered in seeking route recommendation, it is possible to guide more drivers to make better and more informed decisions, e.g., staying in city suburbs to seek for passengers. This not only increases driver revenue, but eases the imbalance between supply and demand and improves service efficiency as well.

To summarize, the reasons why dynamic prices should be considered in seeking route recommendation are two-fold:

- Dynamic price multiplier changes rapidly on the temporal dimension and has a significant impact on driver revenue, so it should be carefully considered in a fine granularity.
- The spatial distribution of dynamic price multipliers reveals some counter-intuitive insights, showing that considering dynamic prices is helpful in guiding drivers to make better and more informed decisions.

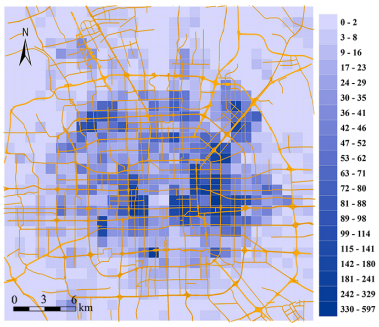


Figure 4: The spatial distribution of the number of orders.

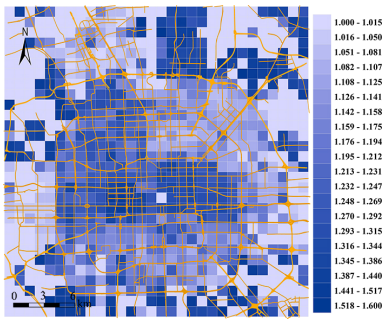


Figure 5: The spatial distribution of dynamic price multipliers.

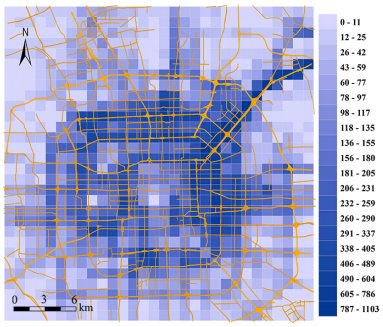


Figure 6: The spatial distribution of the number of vacant cars.

4. Dynamic Price Prediction

The rationale behind doing dynamic price prediction is simple. Firstly, dynamic price multiplier should be considered in seeking route recommendation so as to guide drivers to make better decisions. Secondly, dynamic price multiplier should be evaluated at a fine spatio-temporal granularity, so simply using average or historical price multipliers may not be enough.

The goal of our dynamic price prediction problem is simplified as predicting the dynamic price multiplier given the spatio-temporal information (e.g., time, date, longitude, latitude, etc.). In RoD service, the price multiplier is usually discrete. For example, in our dataset the price multiplier falls in the range $DP_{range} = [1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6]$. Therefore, predicting the price multiplier could be viewed as a classification problem. The input to the classifier is a feature vector extracted from our multi-source urban data that gives further details based on the spatio-temporal information. The output is one of the 7 possible price multipliers. We have the following definition of our dynamic price prediction problem:

Definition 4.1 (Dynamic Price Prediction Problem)**. Given** the spatio-temporal information (e.g., time, date, longitude, latitude, etc.),

Extract an input feature vector, denoted by $\vec{X}$, from our RoD order and dynamic price data, taxi data, POI data and public transport distribution data.

Predict $\hat{p}(y = \bar{d}|\vec{X}), \forall \bar{d} \in DP_{range}$: the probability of a candidate price multiplier $\bar{d}$ being the actual price multiplier y based on the input feature vector $\vec{X}$. The price multiplier with the largest probability is the output of the classification problem. □

4.1. Feature Extraction

In feature extraction, we first obtain the spatio-temporal information (e.g., time, date, longitude and latitude) from every order in our RoD order data, and then generate the following features corresponding to each order. Features are divided into four different contexts, namely temporal context, spatial context, historical dynamic prices context, and taxi service context. The price multiplier obtained from our RoD dynamic price data that corresponds to each order is used as the ground truth in model training.

4.1.1. Temporal Context

Temporal context features are the most basic, and are simply the time and date obtained from each order in RoD order and dynamic price data.

Time feature ET. The hour and minute value (i.e., h and min) are mapped to the radian of a unit circle. On the circle, the 1,440 minutes of a day are represented by a radian value between $[0, 2\pi)$. The time feature ET is defined as:

$$ET = [cos\theta, sin\theta], \theta = \frac{h * 60 + min}{1440} * 2\pi. \quad (2)$$

Date features DM**,** DW **and** DH. DM, DW and DH refer to the day of month, the day of week, and a boolean value describing if the day is a weekday, respectively.

4.1.2. Spatial Context

Features describing the spatial context are extracted mainly from the POI data and public transport distribution data. Based on the longitude and latitude, we describe a location by the POIs and the availability of public transport around it.

POI features POI_n, POI_f, and POI_u. They are all 14-dimension vectors, and reflect the number, frequency and uniqueness of POIs around a location. The 14 dimensions correspond to the 14 POI categories explained in section 3.1.3. We count the POIs within a 500-meter radius of the location. For the i-th category of POI, the number of POIs of this category around the location is denoted by $POI_{n,i}$, and the total number of POIs of this category in the city is M_i, then the frequency and uniqueness of the i-th category of POI – i.e., $POI_{f,i}$ and $POI_{u,i}$ – are defined as:

$$POI_{f,i} = \frac{POI_{n,i}}{\Sigma_{k=1}^{14} POI_{n,k}}, \tag{3}$$

$$POI_{u,i} = \frac{POI_{n,i}}{M_i}. \tag{4}$$

Among these three vectors, POI_n simply counts the number of POIs of different categories around the location; POI_f focuses on the proportion of each category of POI to all POIs around the location; and POI_u characterizes the proportion of the number of a particular category of POIs to the total number of this category in the whole city, which is a reflection of POI uniqueness.

Public transport distribution feature BM. BM is a 4-dimension vector, describing the number of bus stations, the number of bus lines, the number of metro stations and the number of metro lines within a 500-meter radius of the location. It is intuitive that the public transport distribution nearby has impacts on dynamic prices. Firstly, it reflects the popularity of a location. Secondly, the presence of bus and metro stations also makes it possible for RoD drivers to provide connecting services – picking up a passenger who just alighted from a bus or train, or delivering a passenger to a bus or metro station – and hence changes passenger demand around a location.

4.1.3. Historical Dynamic Prices Context

Dynamic price multiplier is a perfect and real-time reflection, as claimed by major RoD service providers, of the supply and demand. As the current supply and demand may be related to the past supply and demand, we hypothesize that the past price multipliers should be helpful in predicting current price multipliers. In other words, historical dynamic prices context is used as a reference for predicting current price multipliers.

Based on the current time on which we want to evaluate the price multiplier, we extract the average price multiplier among all orders taking place within a 500-meter radius of the location, during the last three half-hour-length timeslots. Specifically,

$$HDP = [APM_{-1}, APM_{-2}, APM_{-3}]. \tag{5}$$

In (5), HDP is the historical dynamic prices feature vector, and $APM_{-i} (i = 1, 2, 3)$ are the average price multiplier during the last three timeslots, respectively. For example, if we want to predict the price multiplier at 17:09, then APM_{-1}, APM_{-2} and APM_{-3} are the average price multiplier during $[16 : 30, 17 : 00]$, $[16 : 00, 16 : 30]$, and $[15 : 30, 16 : 00]$, respectively.

4.1.4. Taxi Service Context

As we have explained, taxi GPS trajectories not only serve as a useful guideline for RoD drivers, but characterize the traffic in different spatio-temporal combinations as well. Each feature is calculated on a half-hour-length timeslot basis, and includes the taxis within a 500-meter radius of the location. We extract the following features from taxi GPS trajectories:

- taxi up count UC: the number of taxis that start new orders around the location during the current half-hour-length timeslot.

Table 2: A summary of features used in dynamic price prediction.

Context	Feature	Dim.
Temporal Context	ET: the time feature that maps the hour and minute to the radian of a unit circle	2
	DM: one-hot encoded vector of the the day-of-month value	30
	DW: one-hot encoded vector of the the day-of-week value	7
	DH: a boolean value that equals 1 when the day in question is a weekday	1
Spatial Context	POI_n: the number of POIs, of 14 different categories, around the location	14
	POI_f: the POI frequency vector, as defined in (3), around the location	14
	POI_u: the POI uniqueness vector, as defined in (4), around the location	14
	BM: the number of bus/metro lines and stations around the location	4
Historical Dynamic Prices Context	APM_{-1}: the average price multiplier around the location during the last timeslot	1
	APM_{-2}: the average price multiplier around the location during the second last timeslot	1
	APM_{-3}: the average price multiplier around the location during the third last timeslot	1
Taxi Service Context	UC: taxi up count around the location during the current time slot	1
	DC: taxi down count around the location during the current time slot	1
	VC: taxi visit count around the location during the current time slot	1
	FC: taxi full count around the location during the current time slot	1
	FR: taxi full ratio around the location during the current time slot	1

- taxi down count DC: the number of taxis that terminate existing orders around the location during the current half-hour-length timeslot.
- taxi visit count VC: the number of taxis that present around the location during the current half-hour-length timeslot.
- taxi full count FC: the number of taxis with passengers on board that present around the location during the current half-hour-length timeslot.
- taxi full ratio FR: the ratio of taxi full count FC to taxi visit count VC.

Then the taxi service feature vector is written as,

$$TS = [UC, DC, VC, FC, FR]. \tag{6}$$

4.1.5. The Input Feature Vector

Before creating the input feature vector, it is necessary to normalize the above-mentioned features to guarantee convergence and a shorter training time. Among these features,

- Date features DM and DW are categorical features, and they are represented by one-hot encoding. These features do not need normalization, as components of the one-hot encoded vectors are either 0 or 1.
- Date feature DH is a boolean, and takes the value of either 0 or 1. Hence it is not necessary to normalize DH.
- For all other features, we calculate the Z-score (i.e., the number of standard deviations from the mean) of each feature, or of each component of multi-dimension features, to perform normalization.

For each order, we gather all the above-mentioned features, as summarized in Tab. 2, to generate the input feature vector $\vec{X}$, which are then fed into the dynamic price prediction model:

$$\vec{X} = [ET, DM, DW, DH, POI_n, POI_f, POI_u, BM, APM_{-1}, APM_{-2}, APM_{-3}, UC, DC, VC, FC, FR]. \tag{7}$$

4.2. Model Selection

There are a lot of available algorithms for a predictive task, including the popular deep learning models or more traditional and simpler machine learning models. Though deep learning models have achieved a tremendous progress on voice, text, or image datasets, they may not be good enough for our dynamic price prediction task. As an example, we compare between deep models and tree-based models. Firstly, recent discussions in [39, 40] point out that tree-based models perform better than deep learning models on tabular data. Secondly, deep models are generally more complex and harder to fine tune, sometimes leading to over-fitting problems. The more complex structure of deep models also gives rise to a longer training time. Lastly, deep learning models are usually difficult to interpret or explain, and sometimes sophisticated techniques are needed. By comparison, tree-based models, especially those containing only a small number of trees, are naturally easier to interpret.

We thus choose tree-based models to conduct dynamic price prediction. Specifically, we adopt XGBoost and LightGBM models to predict the dynamic price multipliers, and perform a hit and trial to see which one leads to a higher prediction accuracy. In addition, we also implement a Random Forest model and an ANN (artificial neural network) model to serve as baselines, so that we could compare model performances and justify our model selection considerations. Below we briefly explain XGBoost, LightGBM, and Random Forest.

XGBoost. XGBoost [41] is a widely-used gradient boosting decision tree variant that trains multiple base learners to improve model accuracy. It is highly parallelized and carefully optimized for a shorter training time. It is also currently one of the best, in terms of accuracy and training time, boosting-based tree-models.

LightGBM. LightGBM [42] is another widely-used gradient boosting decision tree variant. But, compared with XGBoost, LightGBM focuses on efficiency and significantly reduces training time and memory consumption by adopting techniques such as EFB (Exclusive Feature Bundling), histogram-based algorithm, leaf-wise tree growth, GOSS (Gradient-based One-Side Sampling), etc. On the other hand, LightGBM is not as robust as XGBoost is, and may be overfitting in some cases.

Random Forest. Random Forest is a bagging-based decision tree variant. Though it appears earlier than XGBoost and LightGBM, it is still widely-used in many real world scenarios. The bagging nature makes it especially suitable for parallel computation and scale well on large-scale high-dimensional datasets. Random Forest samples with replacement from the input data, and also uses feature bagging, or random subspace method, to ensure convergence and low correlation among multiple decision trees.

In section 6 we would compare the performances of XGBoost, LightGBM, Random Forest and ANN, in which the latter two are mainly used as baselines, and provide a detailed discussion on why certain models work better than others. Additionally, we also list the main parameters used in these models, and provide their values that are determined by grid searching.

5. The Reinforcement Learning Model

Now that we have built the dynamic price prediction model, the next task is to perform seeking route recommendation by a reinforcement learning model. Route recommendation could be conducted on different spatial granularities. For example, it could be done on the cell level, i.e., partitioning the area of study into rectangular cells and recommending the next cell a driver should go for seeking after the current cell, or on the road segment level, i.e., recommending the next road segment when a seeking driver comes to a road intersection. Route recommendation on the cell level may have a coarser spatial granularity, but has the advantage of being simple and efficiency, and still giving enough insights.

We choose to study the seeking route recommendation problem on the cell level. Similar to what we have done in data analysis in section 3.2.2, we partition the area of study, i.e., the city of Beijing within longitude $[116.21, 116.56]$ and latitude $[39.81, 40.08]$, into 900 $(= 30 \times 30)$ rectangular cells of equal size. The time range is restricted to a whole day. In other words, we keep a timer τ and set it as $\tau = 0$ in the very beginning, and perform seeking route recommendation until $\tau \geq 1440$ (in minutes). The goal of the reinforcement learning model is to solve the following problem:

Table 3: Notations used in our reinforcement learning model.

Variable	Explanation
s, S	a state such that $s = [l, p, t, e]$, and the set of all states
l	the index of a cell
p	the index of current timeslot
P	the length of each timeslot (in minutes)
N	the total number of timeslots in a whole day
t	the number of minutes into the current timeslot
τ	the timer that represents the current time (in minutes)
e, E	the reverse of the incoming direction, and the set of all directions
a, A	the action a driver takes, and the set of all possible actions
$t_{seek}(j, p)$	the amount of time to seek for passengers in cell j during timeslot p
$d_{seek}(j, p)$	the driving distance to seek for passengers in cell j during timeslot p
$t_{drive}(j, k, p)$	the amount of time to drive from cell j to k during timeslot p
$d_{drive}(j, k, p)$	the driving distance from cell j to k during timeslot p
$P_{pickup}(j, p)$	the probability of picking up a passenger in cell j during timeslot p
$P_{dest}(j, k, p)$	the probability of a passenger picked up in cell j having a destination in cell k during timeslot p
$dp(j, p)$	the predicted dynamic price multiplier at cell j during timeslot p
f_c	the fuel consumption per kilometre
f_{base}	the flag-fall price
f_d	the unit price per kilometre
α	the learning rate in SARSA-λ, $0 < \alpha < 1$
γ	the discount factor of future rewards, $0 < \gamma \leq 1$
ϵ	the probability of random exploration, $0 < \epsilon < 1$
λ	the parameter that controls the fade away speed of past states, $0 < \lambda < 1$

Definition 5.1 (Seeking Route Recommendation)**. Given** the cell division of Beijing, GPS trajectories, RoD order data, and a subset of RoD cars Z.

Find the optimal seeking route for each car in Z to increase earnings. That is, for a single vacant car, when it arrives at a cell, determine which neighboring cell it should go to, or just keep seeking in the current cell. Recommendation is terminated when $\tau \geq 1440$. □

Basically, we solve the seeking route recommendation problem in two steps. In the first step, we use Markov decision process (MDP) to model the environment. In MDP, the transitions between cells follow the Markov property, i.e., the transition probabilities from the current cell to next cells depend only on the current cell and are not related to previous cells. By modelling the environment based on MDP, we are able to determine the reward – i.e., driver earnings – corresponding to cell transitions. The reward is calculated based on the predicted dynamic price multipliers and the possibility of taking up passenger orders. In the second step, based on the environment, we use reinforcement learning to simulate drivers' behavior and determine the optimal seeking routes. The notations used in both two steps are summarized in Tab. 3.

5.1. Modelling the Environment

We use MDP to model the environment. In MDP, there is an agent with a starting state. In every state, the agent chooses one action from many candidates and jumps to another state, getting a reward from the environment that is determined by the combination of state and action. The Markov property states that the state transitions and the corresponding rewards are only dependent on the current state instead of any previous states.

Before we present our MDP model, we first reiterate the spatial and temporal division. Temporally, we divide the 1440 minutes of a whole day into N shorter timeslots, each of which is $P = 1440/N$ minutes long, and we update the parameters of the MDP model in every timeslot. In our study, we let $N = 48$, and hence each timeslot is half an hour. Spatially, as mentioned previously, we partition the city of Beijing within the

longitude range $[116.21, 116.56]$ and latitude range $[39.81, 40.08]$ into 900 ($= 30 \times 30$) rectangular cells of equal size, and each cell is approximately 1 km^2.

In the following, we describe the states, actions, state transitions, and rewards in the MDP model:

States. We denote a state as $s = [l, p, t, e] \in S$. l is the index of a cell and ranges from 1 to 900. p is the index of current timeslot, and ranges from 1 to N. t is the number of minutes in the current timeslot. So, p and t together describe the current time.

e is defined as the reverse of the incoming direction. The incoming direction, as its name suggests, is the direction the driver arrives at the current cell during seeking. The goals of using the reverse of incoming direction are two-fold. Firstly, the whole seeking path is recorded. Secondly, to avoid going into a loop, we require that if a seeking driver does not pick up passengers in the last cell, he or she should not go back to the last cell after the current cell. So, the reverse of the incoming direction is ruled out as a possible choice of action. Specifically, we let $e \in E = \{\searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \uparrow, \nearrow, \rightarrow, \circlearrowleft, \emptyset\}$. Among these directions, $\emptyset$ means "the driver has just dropped off a passenger and there is not a definition of incoming direction", and $\circlearrowleft$ means "the driver has been in this cell in the last state". For simplicity, we also denote these directions as 1 to 10. Fig. 7(a) illustrate these directions. In Fig. 7(a), we assume that the driver is now at the center cell, and the dashed grey arrows represent incoming directions, whereas the blue solid arrows represent the reverse of incoming directions.

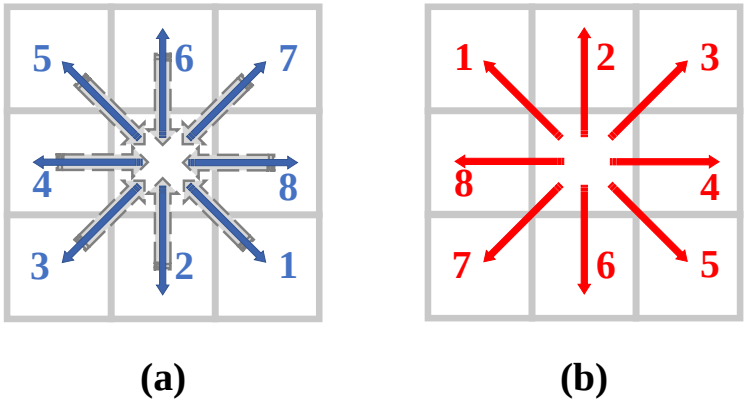


Figure 7: States and actions in the MDP model. Sub-figure (a) shows all possible e, i.e., the reverse of the incoming direction, in a state S. Assuming the driver is in the center cell, and the dashed grey arrows represent the possible incoming directions, so the blue arrows represent the reverse of the incoming directions. Sub-figure (b) shows all possible a, i.e., the actions when the driver is in the center cell.

As an example to explain the definition of a state, a given state $s = [250, 17, 20, \emptyset]$ means that a driver has just dropped off passengers and starts seeking at 8:20am in cell 250.

Actions. At a given state, a driver takes an action and is transitioned to the next state (e.g., jumping to a neighboring cell or keep seeking in the current cell). An action a could be regarded as an outgoing direction. Similarly, we let $a \in A = \{\nwarrow, \uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \circlearrowleft\}$, and also denote these directions as 1 to 9, as shown in Fig. 7(b).

Under the above definitions of e and a, it is clear that we have the following relationship between them. First of all, to avoid going into a loop, at a given state $s_{now} = [l_{now}, p_{now}, t_{now}, e_{now}]$, the actions $a_{unallowed}$ that satisfy the following requirements are not allowed:

$$a_{unallowed} = \begin{cases} e_{now} + 4, & \text{if } 1 \le e_{now} \le 4 \\ e_{now} - 4, & \text{if } 5 \le e_{now} \le 8 \end{cases}. \tag{8}$$

Secondly, if a driver takes an action a and jumps to the next state with the reverse of incoming direction e, then we have $e = a$.

State Transitions. Assuming the driver is at state $s_0 = [i, p, t, e]$, there are two different kinds of state transition after the driver takes an action a and jumps to cell j:

- The driver successfully picks up passengers in cell j. This happens with a probability $P_{pickup}(j,p)$. Then passengers are delivered to their destination, denoted by cell k, with a probability $P_{dest}(j,k,p)$. The time and distance of seeking in cell j are denoted by $t_{seek}(j,p)$ and $d_{seek}(j,p)$, respectively; and the time and distance of delivering passengers from cell j to k are denoted by $t_{drive}(j,k,p)$ and $d_{drive}(j,k,p)$, respectively. The total amount of time from cell i to cell k is $T_{total,1} = t_{drive}(i,j,p) + t_{seek}(j,p) + t_{drive}(j,k,p)$. After the driver delivers passengers in cell k, the state is transitioned to:

$$s_1 = [k, p + (t + T_{total,1})/P, (t + T_{total,1})\%P, \emptyset]. \tag{9}$$

- The driver does not pick up passengers in cell j. This happens with a probability $1 - P_{pickup}(j,p)$. The time and distance of driving from cell i to cell j are $t_{drive}(i,j,p)$ and $d_{drive}(i,j,p)$; and the time and distance of seeking in cell j are $t_{seek}(j,p)$ and $d_{seek}(j,p)$, respectively. The total amount of time from cell i to seeking in cell j is $T_{total,2} = t_{drive}(i,j,p) + t_{seek}(j,p)$. After the driver finishes seeking in cell j, the state is transitioned to:

$$s_2 = [j, p + (t + T_{total,2})/P, (t + T_{total,2})\%P, a] \tag{10}$$

  It should be noted that in (10) the reverse of incoming direction in state s_2 equals a, and we have already explained about this just before we describe state transitions.

Rewards. When a driver is transitioned between two states, a reward is obtained from the environment. We include the impact of dynamic prices into the rewards.

In the first kind of state transition, the rewards consist of two parts: a positive trip fare earned by the driver, and a negative fuel consumption. The fuel consumption could be written as:

$$R_{fuel,1} = -f_c \cdot [d_{drive}(i,j,p) + d_{seek}(j,p) + d_{drive}(j,k,p)]. \tag{11}$$

In (11), f_c is the fuel consumption per kilometre, and $d_{drive}(i,j,p) + d_{seek}(j,p) + d_{drive}(j,k,p)$ is the total distance from state S_0 to S_1. The trip fare could be written as:

$$R_{trip,1} = dp(j,p) \cdot (f_{base} + f_d \cdot d_{drive}(j,k,p)). \tag{12}$$

In (12), f_{base} is the flag-fall price, and f_d is the unit price per kilometre. $dp(j,p)$ is the predicted dynamic price multiplier at cell j, based on our dynamic price prediction model in section 4. Specifically, $dp(j,p)$ is measured at the location in cell j that has the largest number of orders in our data or the center of cell j if such location does not exist.

In the second kind of state transition, as the driver does not pick up passengers in cell j, the reward is only the negative fuel consumption in the following form:

$$R_{fuel,2} = -f_c \cdot [d_{drive}(i,j,p) + d_{seek}(j,p)]. \tag{13}$$

The state transitions and rewards are illustrated in Fig. 8.

5.2. Solving with SARSA-λ

Based on environment, there are generally two different ways to design seeking routes that maximize driver revenue. The first way is dynamic programming, and solves for an optimal policy. But as we use real-time dynamic price prediction, and update the parameters of the MDP model every P minutes, it would become overwhelmingly complicated in dynamic programming. Another way is reinforcement learning, and tries to obtain the optimal state-action pairs that lead to higher rewards in the long run. Comparatively, reinforcement learning is faster but still produces near optimal results on convergence.

SARSA-λ is a typical reinforcement learning algorithm that solves for the optimal state-action pairs. SARSA stands for "*State, Action, Reward, State, Action*" and it works in a similar way to the famous Q-learning algorithm – the driver learns a Q-table that stores a Q-value for each state-action pair describing the utility of taking an action given a state, by trying different actions and observing the rewards returned

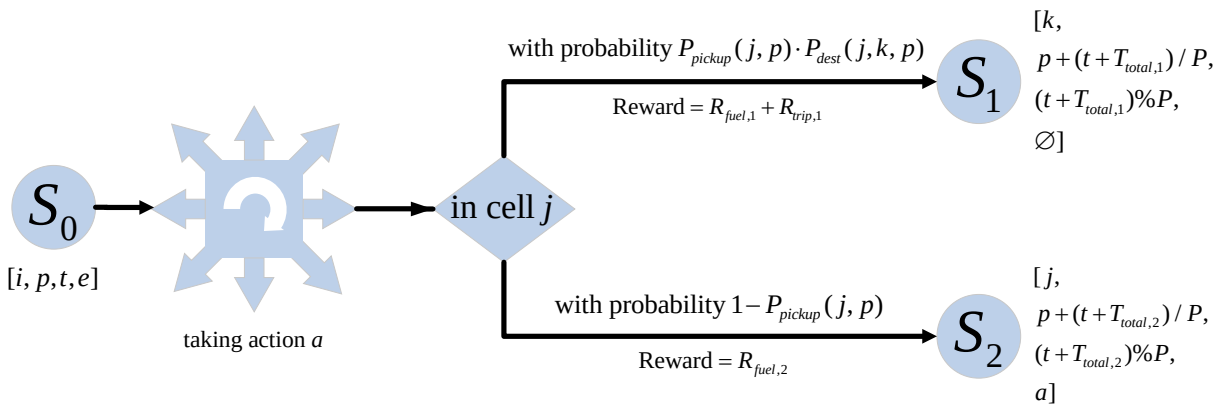


Figure 8: State transitions and rewards.

Algorithm 1 SARSA-λ algorithm

Input:
1. the environment modelled by MDP, λ, ϵ, γ, and α.
2. the dynamic price prediction model in section 4.

Output: the Q-table $Q(s,a)$ for any state-action pair.

$Q(s,a) = 0$ for any s and a; //Initialize Q-table.
$E(s,a) = 0$ for any s and a; //Initialize the eligibility trace E-table.
$p = p_{start}$; //Initialize the timeslot that the driver starts working.
while Q-table not converged **do**
    $\tau = 0$;
    $E(s,a) = 0$ for any s and a;
    Generate a random number $1 \leq l_{init} \leq 900$ that represent the starting cell;
    $s = (l = l_{init}, p = p_{start}, t = 0, e = 0)$; //Initialize state.
    Choose action a at state s using $\epsilon-$greedy policy from Q-table;
    while $\tau < 1440$ **do**
        Take action a and get the predicted dynamic price multiplier $dp(j,p)$;
        Get the new state s' and the reward $R_{s \to s'}$;
        Get the new timer τ';
        Choose action a' at state s' using $\epsilon-$greedy policy from Q-table;
        $\delta \leftarrow R_{s \to s'} + \gamma \cdot Q(s',a') - Q(s,a)$;
        $E(s,a) \leftarrow E(s,a) + 1$; //Record the eligibility trace.
        for all $a \in A$ and $s \in S$ **do**
            $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \delta \cdot E(s,a)$;
            $E(s,a) \leftarrow \gamma \cdot \lambda \cdot E(s,a)$;
        $s \leftarrow s'$, $a \leftarrow a'$, $\tau \leftarrow \tau'$;
return Q-table

from the environment. Unlike Q-learning, SARSA is an on-policy approach. Based on SARSA, SARSA-λ updates the Q-values based on all past states and actions instead of only the current states and actions. To accomplish this, SARSA-λ introduces the eligibility trace by using an E-table and records the whole trace with which a driver has been seeking for.

Algorithm 1 shows the SARSA-λ algorithm for a single driver. The input to the algorithm is the environment modelled by MDP in section 5.1, and the dynamic price multiplier predicted by the our model in section 4; and the output of the algorithm is a Q-table for the driver, so that the driver could choose the

action with the highest Q-value at any given state. The inner loop represents the trial-and-error process of the driver through a whole day, and the outer loop is performing this process for many times until the Q-table converges. In the following, we explain some important lines:

- **Initialization** (line 1 to 3): we first set the Q-table and E-table with all zeros. In addition, we also initialize the timeslot that the driver starts working as p_{start}. For example, $p_{start} = 0$ means we simulate the seeking process from 0:00, whereas $p_{start} = 17$ means the starting time is 8:00 am.
- **Initialization for the outer loop** (line 5 to 8): in every iteration of the outer loop, we initialize the timer $\tau = 0$, the E-table with all zeros, and randomly choose a cell where the driver starts seeking.
- **Exploration v.s. exploitation** (line 9 and 14): at any given state, the driver could either choose exploration (i.e., randomly choosing an action) or exploitation (i.e., choosing the action with the highest Q-value). This is called ϵ-greedy, and ϵ is the probability of exploration. This helps avoiding being stuck at sub-optimal solutions.
- **Incorporating dynamic prices** (line 11 to 12): the price multiplier $dp(j, p)$ is the result of the dynamic price prediction model. The price multiplier is then used in reward calculation.
- **Updating the E-table** (line 16 to 19): in the Q-learning or the original SARSA algorithm, the update of Q-value depends only on the most immediate action. In SARSA-λ, such update takes into account a series of past actions. When $\lambda = 0$, SARSA-λ reduces to SARSA; and when $\lambda = 1$, SARSA-λ remembers all the past actions. Specifically, when the driver takes an action a at a state s, the corresponding $E(s, a)$ is added by 1, and such $E(s, a)$ gradually fades away as time goes by.
- **Updating the Q-table** (line 15 to 19): the update of Q-value is on-policy. The driver takes an action a and is transitioned from state s to s', obtaining a reward $R_{s \to s'}$. Then the driver again uses the ϵ-greedy policy to choose an action a' at state s', and we calculate the TD error δ – the difference between the immediate reward plus the discounted Q-value of (s, a) and the current Q-value estimate. Both δ and $E(s, a)$ are used to generate the new Q-value estimate (as in line 18).

6. Evaluation

We first evaluate our dynamic price prediction model, and then the reinforcement learning model.

6.1. Dynamic Price Prediction

6.1.1. Evaluation Metrics

We use the common metrics – accuracy, precision, recall and F1-score – to evaluate our dynamic price prediction model. These metrics are calculated based on TP (true positive), TN (true negative), FP (false positive) and FN (false negative) samples, as defined below:

$$\begin{aligned} \text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \ \text{Preicison} = \frac{TP}{TP + FP}, \\ \text{Recall} = \frac{TP}{TP + FN}, \ \text{F1} = \frac{2}{1/Precision + 1/Recall}. \end{aligned} \tag{14}$$

As the dynamic price prediction is a multi-class classification problem, we actually use the macro precision, macro recall and macro F1-score in our evaluation, which are the macro average of the above metrics among all N classes:

$$\text{M-Precision} = \frac{\Sigma_{i=1}^{N} \text{Precision}_i}{N}, \ \text{M-Recall} = \frac{\Sigma_{i=1}^{N} \text{Recall}_i}{N}, \ \text{M-F1} = \frac{\Sigma_{i=1}^{N} \text{F1}_i}{N} \tag{15}$$

Table 4: Major hyper-parameters in tree-based models.

Model	**Hyper-parameter**	**Candidates and the Selected Value**	**Explanation**
Random Forest	n_estimators	$[50, 100, \mathbf{200}, 500]$	the number of trees in the random forest
	max_depth	$[5, 10, 20, \mathbf{40}]$	the maximum depth of a tree
	min_samples_leaf	$[1, \mathbf{2}, 5, 10]$	the minimum number of samples required to be at a leaf node
LightGBM	n_estimators	$[100, 500, 1000, \mathbf{2000}, 4000]$	the number of trees (or boosting iterations)
	learning rate	$[0.01, 0.05, \mathbf{0.1}, 0.5]$	the learning (or shrinking) rate
	max_depth	$[2, 5, \mathbf{8}, 12, 20]$	the maximum depth of a tree
	$\alpha_{LightGBM}$	$[0.01, \mathbf{0.1}, 0.2, 0.5]$	the parameter controlling L1-regularization
	$\lambda_{LightGBM}$	$[0.01, 0.1, \mathbf{0.2}, 0.5]$	the parameter controlling L2-regularization
XGBoost	n_estimators	$[100, 500, 100, \mathbf{2000}, 4000]$	the number of trees
	learning rate	$[0.01, 0.05, \mathbf{0.1}, 0.5]$	the learning rate
	max_depth	$[2, 5, 8, \mathbf{12}, 20]$	the maximum depth of a tree
	colsample_bytree	$[0.25, 0.5, \mathbf{0.75}, 1]$	the fraction of features to be used for a tree
	subsample	$[0.25, 0.5, \mathbf{0.75}, 1]$	the fraction of instances to be used for a tree
	$\gamma_{XGBoost}$	$[0.01, 0.05, \mathbf{0.1}, 0.3]$	the minimum split loss parameter controlling over-fitting

6.1.2. Evaluation Setup

As mentioned previously, the total number of orders in our RoD order and dynamic price data is 2,742,120, and we generate an input feature vector for every single order. In our evaluation, we perform two different classification tasks:

- 7-classes prediction: this task has already been explained. As the price multiplier ranges from 1.0 to 1.6, it is natural to have 7 classes in prediction.
- 3-classes prediction: we also divide the price multipliers into three categories, namely the low price multipliers (i.e., 1.0, 1.1 and 1.2), middle price multipliers (i.e., 1.3 and 1.4), and high price multipliers (i.e., 1.5 and 1.6). The goal of 3-classes prediction is to predict which category the price multiplier falls into.

We use XGBoost and LightGBM to predict dynamic price multipliers, and also implement a Random Forest and an ANN model to serve as baselines. We explain the parameters and setup of these four models below:

- ANN: there are two hidden layers that contain 256 and 128 neutrons. The drop-out regularization is added to each hidden layer to control over-fitting. The activation function, output function, loss function are ReLU, softmax and cross-entropy, respectively. Adam optimizer is used. The order data are divided into training set, validation set and test set in a 7:1:2 ratio. The learning rate is set as 0.1 out of four candidates 0.01, 0.05, 0.1 and 0.5. The batch size is set as 64 out of three candidates 32, 64 and 128. These hyper-parameters are selected by grid search.
- Tree-based models: for tree-based models (i.e., XGBoost, LightGBM and Random Forest), we also randomly choose 20% orders as test set, and adopt a 5-fold cross-validation on the remaining 80% orders. Hyper-parameters are also selected by grid search, and some important ones are summarized in Tab. 4. For each hyper-parameter in Tab. 4, the value in bold is selected among candidates.

6.1.3. Evaluation Results

To evaluate the performance of our dynamic price prediction models, we first show the accuracy, macro precision, macro recall and macro F1-score of XGBoost, LightGBM, Random Forest and ANN in both the 7-classes and 3-classes prediction tasks in Tab. 5. We have the following observations:

- XGBoost has the best performance, according to any of the four metrics and in both the 7-classes and 3-classes prediction. Specifically, in the 7-classes prediction, the XGBoost model has an accuracy of 83.82%, and a macro F1-score of 0.8000. The figures for the 3-classes prediction are 90.67% and 0.8532, respectively.

- Tree-based models indeed perform better than the ANN model. This observation holds for both the 7-classes and 3-classes prediction, and even the LightGBM model, which has the weakest performance among tree-based models, is 21.53% better in accuracy than the ANN model in 7-classes prediction.

- The performance improvement in 3-classes prediction compared with 7-classes prediction is much higher in the ANN model than in tree-based models. This is an interesting fact and is more obvious in macro recall and F1-score. We hypothesize that it is the result of poor performance of ANN model on unbalanced datasets. Because dividing price multipliers into 3 classes means different classes are more balanced, the ANN model has a much better performance.

- Comparing between XGBoost, LightGBM indeed has a significantly shorter training time. In our simulation with Intel i7-12700K and GeForce RTX 3070, it takes about 2,500 seconds to converge in XGBoost, whereas the training time is only about 570 seconds in LightGBM.

Table 5: The performances of dynamic price prediction models in both 7-classes and 3-classes tasks.

Model	**Accuracy**		**M-Precision**		**M-Recall**		**M-F1**	
	7-classes	3-classes	7-classes	3-classes	7-classes	3-classes	7-classes	3-classes
XGBoost	0.8382	0.9067	0.8041	0.8608	0.8014	0.8464	0.8000	0.8532
LightGBM	0.7671	0.8672	0.7452	0.8121	0.6662	0.7598	0.6979	0.7812
Random Forest	0.7941	0.8767	0.7440	0.8177	0.7579	0.7901	0.7451	0.8029
ANN	0.6312	0.7843	0.5064	0.6805	0.3895	0.5590	0.3982	0.5858

The above results justify that XGBoost is the most suitable model to perform dynamic price prediction. It should be noted that we value prediction accuracy more than model training time, because the dynamic price prediction model could be off-line and it is acceptable to update and retrain the model every half hour, every hour or even every day.

To further evaluate the performance of the XGBoost model, Tab. 6 and Tab. 7 show the confusion matrix of the 7-classes and 3-classes prediction by XGBoost, respectively. It is clear that:

- The XGBoost has a precision higher than 0.8 for five price multipliers (i.e., 1.0, 1.1, 1.3, 1.5, and 1.6), and has a recall higher than 0.9 for four price multipliers (i.e., 1.0, 1.1, 1.3, and 1.5). This indicates that the XGBoost model has a satisfactory performance for most of the price multipliers.

- Results from Tab. 7 shows that the performance is the best for low price multipliers, followed by high and middle price multipliers. The corresponding F1-score are 0.9484, 0.8494 and 0.7618, respectively. A possible reason is that high and low price multipliers are more regular and predictable, and middle price multipliers are more random. It should also be noted that even with the differences of prediction accuracy among different price multipliers, our XGBoost model still has a satisfactory performance on all these price multipliers.

- Results from Tab. 6 also suggests that our model has a relatively poor performance for price multiplier 1.2 and 1.4, according to recall, precision and F1-score. There are two possible reasons. Firstly, 1.2 and 1.4 are price multiplies standing in the middle, and may be difficult to predict due to a high randomness, as we have already discussed. Secondly, it is observed from Tab. 6 that both price multipliers are always mistakenly predicted as 1.0. This may be the result of an immature pricing algorithm, as our dataset dates back to 2016 when the dynamic pricing mechanism was still at an early developing stage. So in some cases the price multipliers are set to a wrong value, which may be

contradictory to our price prediction model. To improve prediction performance, we need to either collect new data, or find out the circumstances in which the price multipliers 1.2 and 1.4 are more common and then design separate models for such circumstances.

Table 6: The confusion matrix of 7-classes prediction by XGBoost.

Ground truth \ Prediction	1.0	1.1	1.2	1.3	1.4	1.5	1.6	Recall	F1-score
1.0	**244026**	3631	6522	1622	5792	634	2757	0.9209	0.9011
1.1	2914	**64252**	494	306	311	27	134	0.9388	0.9074
1.2	14798	2928	**33702**	1731	4834	231	623	0.5727	0.6437
1.3	1129	366	490	**34397**	1020	108	91	0.9148	0.8713
1.4	10302	1608	3979	2796	**30972**	1375	3479	0.5682	0.6054
1.5	242	44	47	133	491	**20075**	400	0.9367	0.8872
1.6	3211	353	630	368	4393	1375	**32281**	0.7576	0.7837
Precision	0.8822	0.8780	0.7348	0.8318	0.6478	0.8426	0.8118		

Table 7: The confusion matrix of 3-classes (low/mid/high price multipliers) prediction by XGBoost.

Ground truth \ Prediction	low	mid	high	Recall	F1-score
low	**375335**	12767	4168	0.9568	0.9484
mid	19107	**67493**	5511	0.7327	0.7618
high	4792	4829	**54422**	0.8498	0.8494
Precision	0.9401	0.7932	0.8490		

To summarize, the XGBoost model has the best performance among the four models we implement. In the 7-classes and 3-classes prediction, the XGBoost model achieves an accuracy of 83.82% and 90.67%, respectively. Therefore, we adopt the XGBoost model to perform dynamic price prediction, and use the predicted value in our reinforcement learning model that is evaluated in section 6.2.

6.2. Reinforcement Learning Model

In this subsection, we evaluate our reinforcing learning model that use SARSA-λ to perform seeking route recommendation. Our main goals are two-fold:

- We evaluate the effectiveness of introducing dynamic price prediction. Is it necessary to do that? What are the results if average dynamic prices are used instead or no dynamic prices are considered?
- We evaluate the effectiveness of using SARSA-λ. We compare the performance of SARSA-λ with other common reinforcement learning models.

6.2.1. Evaluation Metrics

We define two metrics, namely the revenue efficiency RE and the profit efficiency PE, to evaluate the effectiveness of seeking route recommendation. For a driver, we use R_{total} to denote the total revenue the driver makes during a specified time period; and use T_{total} to denote the total working time during this period (including the time of seeking for and delivering passengers). We also use $T_{deliver}$ to represent the total amount of time used for delivering passengers. Then we have:

$$RE = \frac{R_{total}}{T_{total}} \tag{16}$$

$$PE = \frac{R_{total}}{T_{deliver}} \tag{17}$$

In other words, revenue efficiency RE measures the driver's revenue-making capability comprehensively, whereas profit efficiency PE focuses more on the drivers' ability to find more profitable (i.e., better) orders.

6.2.2. Evaluation Setup

We simulate our SARSA-λ approach based on our RoD order dataset, RoD GPS trajectories and the dynamic price prediction model, to evaluate the effectiveness of seeking route recommendation through SARSA-λ. Across the time range of our datasets, i.e., from August to November 2016, we choose a random Friday and Saturday, as a representative weekday and weekend, to simulate our approach. The chosen days should not be a public holiday (such as the National Day holiday in China). It should also be noted that though we only present results on the chosen Friday and Saturday here, results from other weekdays and weekends show similar effects.

To achieve our main goals, we first learn the parameters of the MDP model, and then randomly choose 500 drivers who work on this Friday or Saturday. The chosen drivers should satisfy the following criteria:

- They work for at least four hours on the chosen day, and their GPS trajectories contain few errors.
- They accept more than one order on the chosen day, and these orders should be effective. For example, orders with very small trip duration or distance may be the result of inaccurate or malfunctioning GPS devices.
- They also work for most of other days.

These criteria ensure that the chosen drivers have regular working patterns, and their trajectories are not outliers. We then adopt a pre-specified model, simulate the trajectories of the chosen drivers, and record the orders, revenue, and trajectories of each driver. The pre-specified model could be the SARSA-λ model, or other variants that serve as baselines, and they would be discussed in section 6.2.3 and 6.2.4 accordingly.

Parameters of the MDP model (i.e., the environment) are set by the following procedures:

- $P_{pickup}(j, p)$: the pickup probability in cell j during timeslot p is approximated by the ratio of the number of orders, denoted by $N_{order}(j, p)$, to the number of vacant cars passing the cell, denoted by $N_{passby}(j, p)$. We only count once if the same vacant car appears continually in a cell during a timeslot.

$$P_{pickup}(j, p) = \frac{N_{order}(j, p)}{N_{passby}(j, p)}. \tag{18}$$

- $P_{dest}(j, k, p)$: the destination probability is approximated by the historical orders. We use $N_{order}(j, k, p)$ to record the total number of orders starting from cell j during timeslot p and ending in cell k, and define $P_{dest}(j, k, p)$ as:

$$P_{dest}(j, k, p) = \frac{N_{order}(j, k, p)}{N_{order}(j, p)}. \tag{19}$$

- $t_{seek}(j, p)$ and $d_{seek}(j, p)$: the seeking distance $d_{seek}(j, p)$ in cell j during timeslot p could be either set as a fixed value – about half of the cell size – or set as a varying value in different timeslots. We try both ways and results show that the difference is small. Hence we set $d_{seek}(j, p)$ to be 500 meter. $t_{seek}(j, p)$ is then set accordingly, based on the average driving speed in cell j during timeslot p.
- $t_{drive}(j, k, p)$ and $d_{drive}(j, k, p)$: the driving time and distance starting from cell j during timeslot p to cell k are approximated by the average driving time and distance in our historical orders. If, for some (j, k, p) combination the number of historical orders is zero, then we resort to the AMap API to check for the estimated driving time and distance instead.
- f_{base}, f_d and f_c: f_{base} and f_d are set to 15 and 2.8 (both in RMB Yuan) according to the service provider's policy. f_c is set to 0.5, similar to previous studies (e.g., [27]).

Hyper-parameters of the SARSA-λ model are also selected based on grid search, similar to what we do in section 6.1.2. We briefly give the choice of some major hyper-parameters below:

- α: the learning rate α is set to 0.1 among candidates $[0.03, 0.05, 0.1, 0.2, 0.3]$.
- γ: the discount factor of future rewards γ is set to 0.5 among candidates $[0.1, 0.3, 0.5, 0.7]$.
- ϵ: the probability of random exploration ϵ is set to 0.1 among candidates $[0.05, 0.1, 0.3, 0.5, 0.7]$.
- λ: the parameter controlling the impacts of past states ϵ is set to 0.5 among candidates $[0, 0.1, 0.3, 0.5, 0.7, 1]$.

6.2.3. The Effectiveness of Dynamic Price Prediction

To evaluate the effectiveness of dynamic price and dynamic price prediction, our simulation compares the SARSA-λ model with the ground-truth and two baselines:

- **Real**: the ground-truth from our data;
- **SL-DPP**: our SARSA-λ model explained in section 5.2. SL-DPP stands for "SARSA Lambda-Dynamic Price Prediction".
- **SL-ADP**: SL-ADP is similar to SL-DPP. But instead of using dynamic price prediction in calculating rewards, SL-ADP uses the average historical dynamic price multiplier in a cell. SL-ADP stands for "SARSA Lambda-Average Dynamic Prices".
- **SL-1.0**: SL-1.0 is similar to SL-DPP, but it ignores dynamic prices at all. This is equivalent to setting all $dp(j, p)$ – the price multiplier in cell j during timeslot p – to 1.0.

We first present the distribution of revenue efficiency RE, profit efficiency PE, and searching time (the amount of time used to seek for passengers) of SL-DPP, SL-ADP and SL-1.0 on four selected time periods on weekday. The selected time periods are [8:00, 9:00), [12:00, 13:00), [18:00, 19:00) and [23:00, 0:00), and cover representative hours such as morning and evening rush hours, night hours, and non-rush hours around noon. The distribution of revenue efficiency, profit efficiency and searching time are shown in boxplots in Fig. 9, Fig. 10, and Fig. 11, respectively. Tab. 8 further gives the average and median values of these metrics on the selected periods on weekday.

Table 8: The average and median of revenue efficiency, profit efficiency and searching time on weekday.

Period	**Scheme**	**revenue efficiency**		**profit efficiency**		**searching time**	
		average	median	average	median	average	median
8:00	Real	0.88	0.90	1.11	1.11	15.49	13.00
	SL-1.0	1.00	1.01	1.15	1.15	10.05	8.00
	SL-ADP	1.31	1.32	1.49	1.49	9.04	7.50
	SL-DPP	1.45	1.45	1.82	1.82	14.93	13.00
12:00	Real	0.80	0.80	1.18	1.15	22.63	22.00
	SL-1.0	1.01	1.01	1.24	1.22	13.70	12.00
	SL-ADP	1.16	1.14	1.42	1.38	13.40	12.00
	SL-DPP	1.22	1.22	1.58	1.56	15.91	15.00
18:00	Real	0.81	0.80	1.12	1.11	19.73	18.00
	SL-1.0	1.01	0.99	1.22	1.18	12.09	10.00
	SL-ADP	1.33	1.32	1.59	1.56	11.65	10.00
	SL-DPP	1.49	1.49	1.80	1.76	11.95	10.00
23:00	Real	0.78	0.73	1.31	1.29	28.14	28.00
	SL-1.0	1.01	1.01	1.38	1.37	18.88	18.00
	SL-ADP	1.25	1.25	1.70	1.67	18.85	17.00
	SL-DPP	1.30	1.29	1.82	1.82	20.32	18.00

Fig. 9 to 11, together with Tab. 8, indicate that:

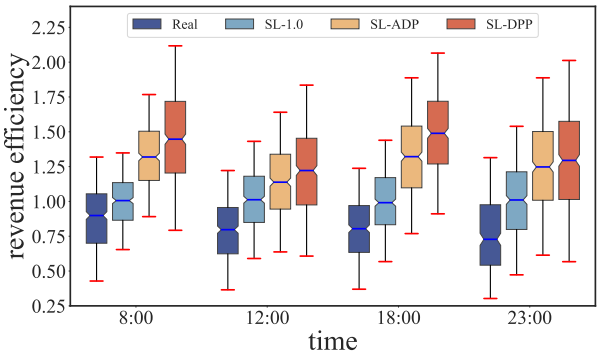


Figure 9: The distribution of revenue efficiency on selected periods on weekday.

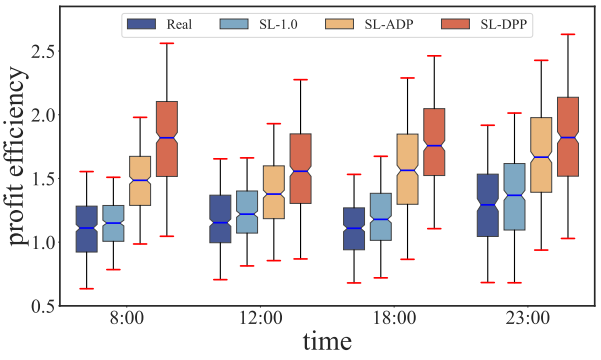


Figure 10: The distribution of profit efficiency on selected periods on weekday.

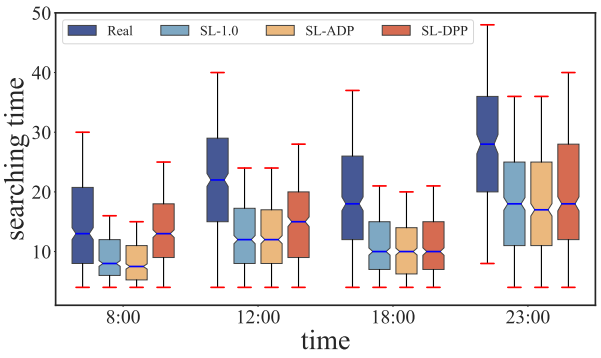


Figure 11: The distribution of searching time on selected periods on weekday.

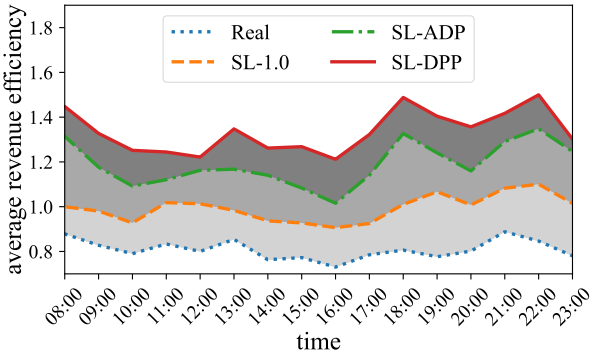


Figure 12: The average revenue efficiency throughout the whole day (weekday).

- **Using the SARSA-λ reinforcement learning model is effective**. In all four selected periods, even the SL-1.0 scheme improves the seeking efficiency than ground-truth. For example, during [8:00, 9:00), comparing between SL-1.0 and ground-truth, the average revenue efficiency and profit efficiency among all drivers are increased by 13.6% and 3.6%, and the average searching time is reduced by about 35%. This indicates that reinforcement learning is able to help drivers to find better orders and get orders quickly.

- **It is necessary to consider dynamic prices in seeking route recommendation**. In all four selected periods, the revenue efficiency and profit efficiency are higher with SL-ADP and SL-DPP than with ground-truth or SL-1.0. For example, during [18:00, 19:00), the revenue efficiency in ground-truth and with SL-1.0, SL-ADP and SL-DPP are 0.81, 1.01, 1.33 and 1.49, respectively. In other words, the two schemes SL-ADP and SL-DPP that consider dynamic prices increase revenue efficiency by 31.6% and 47.5% compared to SL-1.0, respectively.

- **Using predicted dynamic prices further increases seeking efficiency than simply using average statistics**. In all four selected periods, SL-DPP achieves higher revenue and profit efficiency than SL-ADP. Taking the revenue efficiency as an example, SL-DPP gives a revenue efficiency 10.7%, 5.2%, 12.0% and 4.0% higher than SL-ADP does, during [8:00, 9:00), [12:00, 13:00), [18:00, 19:00) and [23:00, 0:00), respectively. We also learn from these figures that the increase of revenue efficiency in morning and evening rush hours is higher than in other two periods. Dynamic price predictions prove to be effective in capturing the rapidly fluctuating dynamic prices, which is beyond the capability of using average statistics.

- **Reinforcement learning reduces searching time, but considering dynamic prices goes to the opposite direction**. This is an interesting observation. It is clear from Fig. 11 that SL-DPP usually leads to a higher searching time than other schemes, whereas SL-1.0 always has a lower searching time compared with ground-truth. We consider the reason is that it takes more time for drivers to find higher price multipliers. Despite a longer searching time, drivers indeed find more profitable orders by considering dynamic prices, especially the predicted dynamic prices.

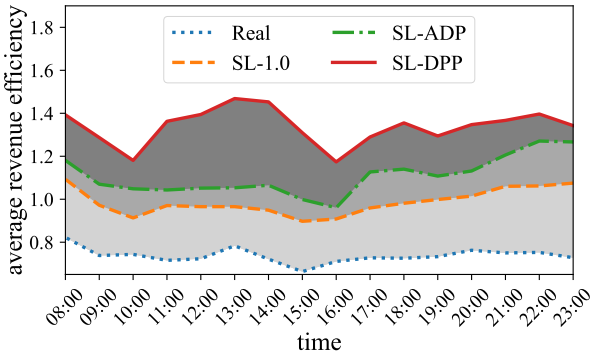


Figure 13: The average revenue efficiency throughout the whole day (weekend).

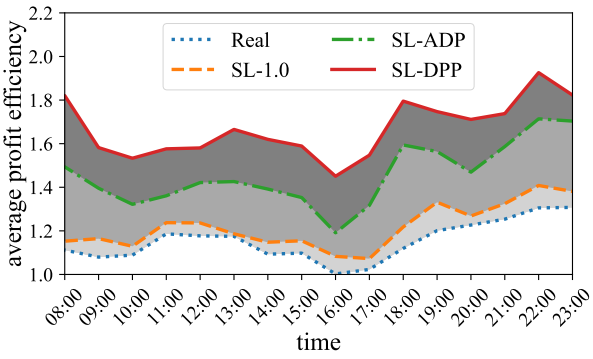


Figure 14: The average profit efficiency throughout the whole day (weekday).

We then evaluate model performances on different hours and on weekend. Fig. 12 and Fig. 13 plot the average revenue efficiency throughout the day on weekday and weekend, respectively. Fig. 14 plots the average profit efficiency throughout the day on weekday. We have the following observations:

- Our above insights about the effects of using reinforcement learning, considering dynamic prices or dynamic price predictions hold throughout the day, either on weekday or weekend. In other words, SL-DPP gives the highest revenue, followed by SL-ADP, SL-1.0 and then the ground-truth.
- The amount of increase in revenue efficiency varies in different hours-of-day on weekday and weekend. For example:
  - SL-DPP v.s. SL-ADP on weekday: the increase is the highest during [13:00, 16:00), which is a time period with relatively stable and lower price multipliers. This shows that considering dynamic price predictions is especially important when the price multipliers across the city is lower and stable, because dynamic price prediction is an extra source of information other than the average statistics.
  - SL-DPP v.s. SL-1.0 on weekday: the increase is the highest during the morning and evening rush hours. This indicates that using dynamic prices – even the average statistics – helps drivers to capture the supply and demand fluctuation in busy and high demand periods and thus increase driver revenue.
  - SL-DPP v.s. SL-ADP or SL-1.0 on weekend: the increase is the highest during [11:00, 14:00). Note that passenger demand pattern on weekend is different from that on weekday, and the number of orders peak around noon or early afternoon [38]. And according to Fig. 3, the price multiplier is relatively low and stable during this period. This observation shows again that dynamic price multiplier is helpful as an extra source of information.
- Comparing between Fig. 14 and Fig. 12 further verifies our previous observations on profit and revenue efficiency. For example:
  - Throughout the day, reinforcement learning helps drivers earn more by reducing the searching time, but the quality of orders largely remains the same. This is clear by inspecting the profit

efficiency and revenue efficiency of SL-1.0 and ground-truth. From Fig. 12 it is shown that driver indeed earn more in SL-1.0 than in ground-truth. But Fig. 14 shows that the profit efficiencies of these two schemes are very close, indicating that drivers obtain similar orders.

– Further considering the dynamic prices, regardless of the averages or predictions, improves both driver revenue and order quality. In both Fig. 12 and Fig. 14, either SL-DPP or SL-ADP significantly increases the profit and revenue efficiency throughout the day than the ground-truth. It also proves that when considering dynamic prices, the improvement of order quality is high enough to compensate for the loss introduced by a longer searching time.

6.2.4. The Effectiveness of Using SARSA-λ

Similar to section 6.2.3, to evaluate the effectiveness of using SARSA-λ, our simulation compares the SL-DPP model with the following two baselines:

- **Q-DPP**: Q-DPP is similar to SL-DPP, as it uses dynamic price prediction. The difference is that Q-DPP uses Q-learning to recommend seeking routes instead of SARSA-λ. Q-learning is also a common reinforcement learning model based on Q-table, but it is off-policy. We also set $\epsilon = 0.1$, as in our SARSA-λ model.
- **DRL-DPP**: DRL-DPP is also similar to SL-DPP and Q-DPP, but it adopts a deep reinforcement learning model to recommend seeking routes, instead of SARSA-λ or Q-learning. Specifically, deep Q-networks replace the Q-table: the input to the Q-network is the current state information, and the output is the Q-value. The deep reinforcement learning model estimates the Q-values by training a deep learning model, and it also adopts mechanisms such as experience replay and target Q-network to improve its performance.

We present the distribution of revenue efficiency and profit efficiency of SL-DPP, Q-DPP and DRL-DPP on the four selected periods on weeday, in Fig. 15 and Fig. 16, respectively. We also show the average revenue efficiency and profit efficiency throughout the whole day, in Fig. 17 and Fig. 18. It is clear from these figures that:

- **Compared to the ground-truth, SL-DPP, Q-DPP and DRL-DPP all significantly increase driver earnings; and SL-DPP indeed has the best performance throughout the day**. For example, during [18:00, 19:00), the average revenue efficiency of SL-DPP, Q-DPP and DRL-DPP is 1.49, 1.39 and 1.30, respectively; and the average profit efficiency of SL-DPP, Q-DPP and DRL-DPP is 1.80, 1.69 and 1.59, respectively. Our hypothesis is that SL-DPP performs the best because it adopts the E-table to record historical traces and this helps learning good policies. For the reason why DRL-DPP does not has a satisfactory performance, we give a more detailed discussion in section 7.
- **The difference between SL-DPP and Q-DPP (or DRL-DPP) differs obviously during different hours-of-day.** For example, during the time periods with more severe supply-demand imbalance (e.g., evening rush hours), SL-DPP has a higher increase of revenue efficiency. We are inspired by this and the last observation that SL-DPP is more capable to capture the high quality orders resulting from supply-demand imbalance and thus higher price multipliers.

Furthermore, to evaluate the effectiveness of using SARSA-λ on weekend, Fig. 19 shows the average revenue efficiency of SL-DPP, Q-DPP and DRL-DPP on weekend, as a comparison to Fig. 17. Fig. 20 compares the revenue efficiency of SL-DPP on weekday and weekend. Because of the limited space, we list some key observations below:

- The above observations about the performance of SL-DPP, Q-DPP and DRL-DPP also hold throughout the day on weekend.
- Regarding the SL-DPP model, the revenue efficiencies on weekday and weekend show different patterns. For example, during morning or evening rush hours, the revenue efficiency is higher on weekday; during [11:00, 15:00), i.e., noon and early afternoon, the revenue efficiency is higher on weekend.

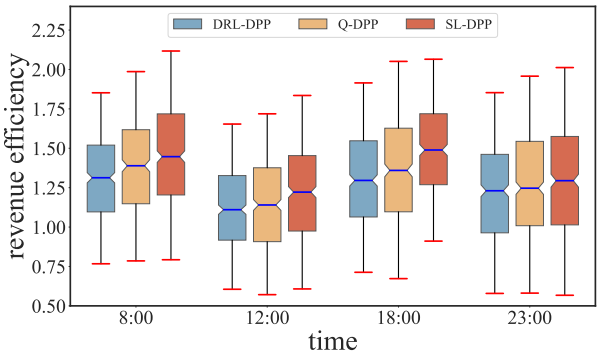


Figure 15: The distribution of revenue efficiency of SL-DPP, Q-DPP and DRL-DPP on selected periods on weekday.

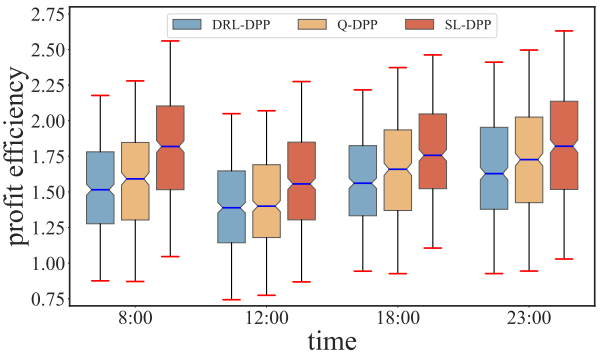


Figure 16: The distribution of profit efficiency of SL-DPP, Q-DPP and DRL-DPP on selected periods on weekday.

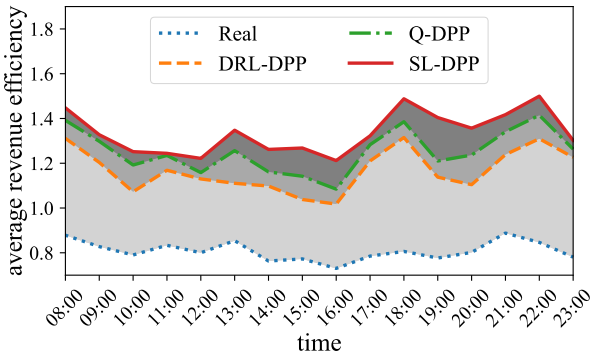


Figure 17: The average revenue efficiency of SL-DPP, Q-DPP and DRL-DPP throughout the whole day (weekday).

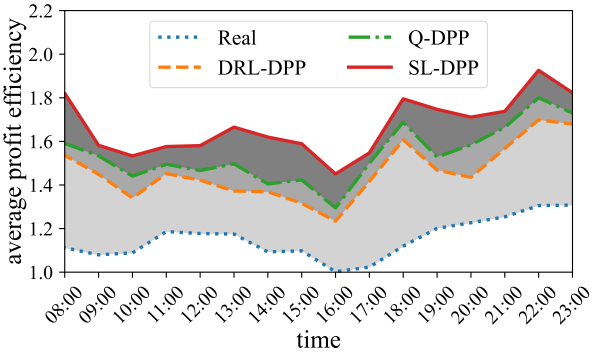


Figure 18: The average profit efficiency of SL-DPP, Q-DPP and DRL-DPP throughout the whole day (weekday).

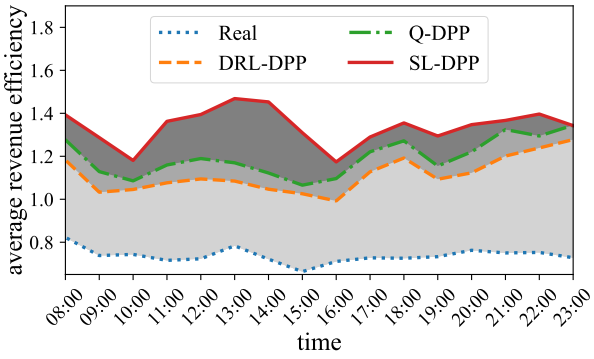


Figure 19: The average revenue efficiency of SL-DPP, Q-DPP and DRL-DPP throughout the whole day (weekend).

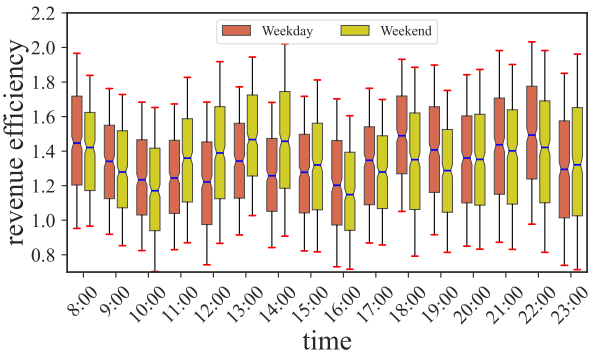


Figure 20: The distribution of revenue efficiency of SL-DPP throughout the whole day (weekday v.s. weekend).

7. Summary and Discussions

We give a brief summary based on our evaluation results, and provide relevant discussions. Our study focuses on using both reinforcement learning and dynamic price prediction to provide seeking route recommendations to drivers, and we thus summarize the effects of considering dynamic price prediction, and using reinforcement learning, respectively. Furthermore, the comparison of different reinforcement learning models in section 6.2.4 gives some intriguing results, and it is necessary to include a further discussion. Finally, we discuss how dynamic prices help avoiding recommending nearby drivers to same locations, which

is a common problem with single agent reinforcement learning models.

The effects of considering dynamic price prediction. In fact, regarding dynamic prices, we compare three different circumstances: considering no dynamic prices, using only the average dynamic price multipliers, and using the predicted dynamic price multipliers. Among these three circumstances, our evaluation results show that the revenue efficiency increases progressively.

Besides, we also summarize the following findings from evaluation results:

- Considering dynamic prices, no matter the average statistics or the predicted price multipliers, leads to a significant improvement of order quality and thus increases driver revenue.
- Considering dynamic prices, especially the predictions, increases the searching time, i.e., drivers need more time to search for better orders. But the improvement of order quality could offset the longer searching time, and thus driver revenue is further increased, compared to only using the average dynamic prices.
- For busy periods with high demand, dynamic price multipliers, even the average ones, help drivers to capture the supply and demand fluctuation and thus obtain higher revenues.
- For periods with generally lower price multipliers and more stable demand, the predicted dynamic price multipliers are especially important. The predicted price multipliers serve as an extra source information and help drivers to find out locations where there are better orders.

The effects of using reinforcement learning. In our study, the effects of using reinforcement learning mainly come from two aspects. Firstly, an agent in SARSA-λ measures the utilities of taking different actions by considering long-term rewards, i.e., looking ahead. Secondly, SARSA-λ remembers the whole trace with the variable λ, i.e., looking backwards. By trail-and-error, the agent finally learns the optimal state-action pairs that maximize the expectation of the sum of rewards. Our evaluation results verify and show the following key findings:

- Even without considering dynamic prices in any form, reinforcement learning alone could already help drivers to find better orders and get orders quickly. For example, our results show that during [8:00, 9:00), the average revenue efficiency is increased by 13.6%, and the average searching time is reduced by about 35%, when only SARSA-λ is used and no dynamic prices are considered.
- The effects of reinforcement learning are, to some extent, in an opposite direction against the effects of dynamic prices; but combining them together yields better results. Specifically, considering dynamic prices means better orders and longer searching time, whereas using reinforcement learning means better orders and shorter searching time. Among these effects, shorter searching time is more obvious in using reinforcement learning, and better orders is more obvious in considering dynamic prices. When both reinforcement learning and dynamic pricing are adopted, these effects offset each other to some extent and driver revenue is further increased.
- We also consider multiple reinforcement learning models, and even though their performances differ, any one of them could bring a significant improvement of revenue efficiency compared to ground-truth.

The comparison between different reinforcement learning models. Our evaluation results show that among the three chosen reinforcement learning models, SARSA-λ (SL-DPP) has the best performance, followed by Q-learning (Q-DPP), and then deep Q-network (DRL-DPP), and this observation holds throughout the day, either on weekday or weekend. This observation is intriguing in that deep Q-network has the worst performance, and in the following we make a comparison between them.

Firstly, deep Q-network is more suitable with high-dimensional or continuous state space, and since our problem does not involve a high-dimensional state space, the advantage of deep Q-network is already diminished. Basically, for high-dimensional or continuous state space, storing the Q-table costs a large amount of storage, making Q-learning or SARSA-λ unrealistic. For deep Q-network, only the parameters of the involved networks are stored, leading to storage space savings. In our study, we could train our SARSA-λ

model for each timeslot so that the state space size is reduced, and it takes less than 2GB memory. Therefore, the problem in our study does not require using deep Q-network to solve.

Secondly, to adopt to high-dimensional or continuous state space problems, deep Q-network uses neural networks to output Q-values. This, in fact, trades the accuracy of Q-values for running time and storage space.

Thirdly, the training of deep Q-network is more complicated, as explained below:

- Fine-tuning requires more efforts. We find that varying some hyper-parameters may lead to drastic performance degradation, and sometimes the training may not even converge. In our evaluations, we already give the best results, but it still could not outperform SARSA-λ and Q-learning. This also shows the difficulty of fine-tuning.
- The training time is also longer. We normally train for 5,000 to 10,000 epochs before convergence. The training time for SARSA-λ and Q-learning is about 10 to 15 minutes, whereas the training time for deep Q-network ranges between 50 minutes to one hour.
- The training of deep Q-network requires sophisticated hardware such as high-end GPUs. By comparison, only an ordinary CPU is needed in training SARSA-λ and Q-learning.

Lastly, SARSA-λ and Q-learning are more light-weight, whereas deep Q-network is more heavy-weight. Specifically, in SARSA-λ and Q-learning, Q-values are stored as tables, and recommending seeking routes means reading from these tables, costing little memory and computation resources. To the opposite, in deep Q-network, either training the neural networks or calculating neural network outputs requires a lot of computation resources. Therefore, in future deployment, it would be possible to deploy the SARSA-λ or Q-learning models to drivers' cell phones or on-car mobile devices, but it would be highly improbable for deep Q-network.

The above discussions mainly focus on the advantages of SARSA-λ or Q-learning over deep Q-network, but it should be noted that deep Q-network has some important advantages, which may be less obvious on low or mid-dimensional problems such as ours. For example:

- Deep Q-network could handle continuous or high-dimensional state space, as mentioned previously.
- By using the experience replay mechanism, deep Q-network store and reuse samples, and could thus effectively handle the data sparsity problem.
- By using neural networks to approximate Q-values, deep Q-network is capable to learn complex strategies and value functions and could thus better adapt to complex environment and tasks.
- Deep Q-network supports end-to-end learning. By using deep neural networks, it is able to learn from original inputs such as images or trajectories instead of manually-designed features.

To sum up, normally we may consider that deep Q-network is better than SARSA-λ or Q-learning. In fact, it would be more accurate to claim that deep Q-network could handle more complex situations and achieve a satisfactory performance in such situations. Yet when it comes to low or mid-dimensional state space problems such as ours, deep Q-network may lose its advantages in performance due to reasons such as inaccurate outputs approximated by neural networks, highly demanding fine-tuning, longer training time, and etc.

The role of dynamic prices in avoiding similar or same recommendations. A common problem arising in similar single-agent studies is that similar or same recommendations would be possibly made to drivers nearby (e.g., in the same cell) or drivers with similar properties. Such recommendations are undesirable, as drivers may then concentrate in a very small area, leading to a more-than-enough supply and unpredictably-fluctuating dynamic price multipliers. Even multi-agent reinforcement learning models are not good enough in solving this problem, as one needs a lot of parameters in order to formulate accurate models, such as drivers' emotional state, drivers' adoption rate of the recommended routes, changes to the environment (e.g., all those probabilities) after driver adoption, etc. These parameters are, unfortunately,

not easy to obtain without laborious and arduous field tests and collaboration from real service providers. We thus consider multi-agent models are not reliable and realistic enough until, in foreseeable future, when such field tests become possible. There are also some heuristics to tackle problem, such as generating a list of possible recommendations (that are equally or almost equally optimal) and randomly picking one after another for drivers nearby.

The adoption of dynamic prices in RoD service offers a new perspective in solving this problem. For example, if the pricing algorithm is perfectly designed and could respond to the changes of supply and demand in real time, then the price multiplier within a small "hot" area would go down when drivers begin to gather there. This lowers the rewards of going to this "hot" area, and keeps other drivers away. With the adoption of dynamic prices, even single-agent models such as ours could avoid recommending a particular cell to a lot of drivers. In our study, on one hand, we assume that the dynamic pricing mechanism of the service provider from which we obtain our datasets satisfies the above requirement, but how to design such a mechanism is another story and is out of the scope of this paper. On the other hand, our inclusion of a dynamic price prediction model could be regarded as a way to imitate the service provider's dynamic pricing mechanism: such model provides a description of the environment at a finer spatial and temporal granularity, compared with the traditional descriptions by the pickup probabilities or destination probabilities.

8. Conclusion

We focus on the seeking route recommendation problem, i.e., recommending the next cell to a seeking driver so that driver revenue is higher, in RoD service. Though this problem has been studied from various perspectives in taxi service, RoD has two new features – dynamic pricing and data-driven – that enable us to improve driver revenue to the next level.

By analyzing real service data, we point out that it is necessary to both recommend seeking routes to drivers and take into account dynamic price multipliers in generating recommendations. We first design a dynamic price prediction model to generate the predicted price multiplier given the time and location described by features from multi-source urban data. We then adopt a reinforcement learning model, and calculate the rewards of transitioning between cells based on the predicted price multipliers.

Evaluation results first validate the effectiveness of both models. The dynamic price prediction model achieves a satisfactory accuracy of 83.82% and 90.67%, in the 7-classes prediction (i.e., predicting the exact multiplier) and 3-classes prediction (i.e., predicting whether the multiplier is high, mid, or low), respectively. The reinforcement learning model also significantly increases drivers' average revenue as well as profit efficiency and reduces average searching time. We also validate the positive effects of considering dynamic price predictions: using predicted price multipliers is better than using average price multipliers, which is better than considering no dynamic prices at all, in terms of both revenue and profit efficiency.

Besides effectiveness validation, our results also reveal some interesting facts. For example, using reinforcement learning primarily reduces searching time, whereas considering dynamic prices as well as predictions mainly enables drivers to improve order quality, though meanwhile the searching time becomes longer. This shows the different roles reinforcement learning and dynamic prices play in increasing driver revenue. We also find out that dynamic price prediction has positive effects because it serves as a new and reliable source of information and captures the supply and demand fluctuation. Furthermore, by comparing between multiple reinforcement learning models, simple models such as SARSA-λ and Q-learning prove to have better performance than complex models such as deep reinforcement learning.

For future work, we primarily consider the implementation on real service. There are many details and perspectives to take into consideration. For example, we would like to study the effects of multi-agent models. In using such models, as we mention previously, there are a lot of information to collect – e.g., the adoption ratio, the changes of passenger pattern after drivers take new recommended routes, the interaction between drivers, etc. – and we are working actively to push the collaboration with service providers to make it happen. It is also necessary to study the interactive effects when multi-agent models are coupled with dynamic prices. Another interesting topic would be implementing our models as a mobile app or an on-car device that automatically guides a driver throughout the working shift.

References

[1] A. Picchi, Uber vs. Taxi: Which Is Cheaper?, 2016. URL: http://bit.ly/2DMgrMc.

[2] Y. M. Nie, How can the taxi industry survive the tide of ridesourcing? evidence from shenzhen, china, Transportation Research Part C: Emerging Technologies 79 (2017) 242–256.

[3] L. Rayle, D. Dai, N. Chan, R. Cervero, S. Shaheen, Just a better taxi? a survey-based comparison of taxis, transit, and ridesourcing services in San Francisco, Transport Policy 45 (2016) 168–178.

[4] J. D. Hall, C. Palsson, J. Price, Is Uber a substitute or complement for public transit?, 2017. URL: https://bit.ly/2K6Vs7L.

[5] T. Berger, C. Chen, C. B. Frey, Drivers of disruption? estimating the uber effect, European Economic Review 110 (2018) 197–210.

[6] Y. Cao, T. S. Gruca, B. R. Klemz, Internet pricing, price satisfaction, and customer satisfaction, International Journal of Electronic Commerce 8 (2003) 31–50.

[7] M. L. Kasavana, A. J. Singh, Online auctions, Journal of Hospitality & Leisure Marketing 9 (2001) 127–140.

[8] M. D. Wittman, P. P. Belobaba, Dynamic pricing mechanisms for the airline industry: a definitional framework, Journal of Revenue and Pricing Management 18 (2019) 100–106.

[9] J. M. Betancourt, A. Hortasu, A. Oery, K. R. Williams, Dynamic Price Competition: Theory and Evidence from Airline Markets, Working Paper 30347, National Bureau of Economic Research, 2022. URL: http://www.nber.org/papers/w30347. doi:10.3386/w30347.

[10] O. Besbes, A. Zeevi, Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms, Operations Research 57 (2009) 1407–1420.

[11] J. Hall, C. Kendrick, C. Nosko, The effects of Uber's surge pricing: a case study, 2015. URL: http://bit.ly/2kayk9O.

[12] J. Gan, B. An, H. Wang, X. Sun, Z. Shi, Optimal pricing for improving efficiency of taxi systems., in: Proceedings of the 22th International Joint Conferences on Artificial Intelligence, IJCAI '13, AAAI, 2013, pp. 2811–2818.

[13] L. Rayle, S. Shaheen, N. Chan, D. Dai, R. Cervero, App-based, on-demand ride services: Comparing taxi and ridesourcing trips and user characteristics in San Francisco, 2014. URL: http://bit.ly/2kVkahg.

[14] L. Chen, A. Mislove, C. Wilson, Peeking beneath the hood of Uber, in: Proceedings of the 2015 ACM Conference on Internet Measurement Conference, IMC '15, ACM, New York, NY, USA, 2015, pp. 495–508.

[15] S. Guo, C. Chen, Y. Liu, K. Xu, D. M. Chiu, Modelling passengers' reaction to dynamic prices in ride-on-demand services: A search for the best fare, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 1 (2018) 136:1–136:23.

[16] S. Guo, Y. Liu, K. Xu, D. M. Chiu, Understanding ride-on-demand service: Demand and dynamic pricing, in: Pervasive Computing and Communication Workshops (PerCom Workshops), 2017 IEEE International Conference on, IEEE, 2017, pp. 509–514.

[17] H. Chen, Y. Jiao, Z. Qin, X. Tang, H. Li, B. An, H. Zhu, J. Ye, InBEDE: Integrating contextual bandit with TD learning for joint pricing and dispatch of ride-hailing platforms, in: Proceedings of the 2019 IEEE International Conference on Data Mining, ICDM '19, 2019, pp. 61–70.

[18] M. K. Chen, 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, EC '16, ACM, New York, NY, USA, 2016, pp. 455–455.

[19] P. Cohen, R. Hahn, J. Hall, S. Levitt, R. Metcalfe, Using big data to estimate consumer surplus: The case of uber, 2016. URL: http://bit.ly/2pqXiWo.

[20] S. Guo, C. Chen, Y. Liu, K. Xu, D. M. Chiu, It can be cheaper: Using price prediction to obtain better prices from dynamic pricing in ride-on-demand services, in: Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous '17, ACM, 2017, pp. 146–155.

[21] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, D. M. Chiu, Fine-grained dynamic price prediction in ride-on-demand services: Models and evaluations, Mobile Networks and Applications 25 (2020) 505–520.

[22] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, D. M. Chiu, Dynamic price prediction in ride-on-demand service with multi-source urban data, in: Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous '18, ACM, 2018, pp. 412–421.

[23] S. Guo, C. Chen, Y. Liu, K. Xu, B. Guo, D. M. Chiu, How to pay less: a location-specific approach to predict dynamic prices in ride-on-demand services, IET Intelligent Transport Systems 12 (2018) 610–618.

[24] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, D. Zhang, D. M. Chiu, A simple but quantifiable approach to dynamic price prediction in ride-on-demand services leveraging multi-source urban data, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2 (2018) 112:1–112:24.

[25] B. Li, D. Zhang, L. Sun, C. Chen, S. Li, G. Qi, Q. Yang, Hunting or waiting? discovering passenger-finding strategies from a large-scale real-world taxi dataset, in: Pervasive Computing and Communication Workshops (PerCom Workshops), 2011 IEEE International Conference on, IEEE, 2011, pp. 63–68.

[26] D. Zhang, L. Sun, B. Li, C. Chen, G. Pan, S. Li, Z. Wu, Understanding taxi service strategies from taxi gps traces, IEEE Transactions on Intelligent Transportation Systems 16 (2015) 123–135.

[27] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, Z. Yu, D. Zhang, D. M. Chiu, ROD-Revenue: Seeking strategies analysis and revenue prediction in ride-on-demand service using multi-source urban data, IEEE Transactions on Mobile Computing 19 (2020) 2202–2220.

[28] H. Rong, X. Zhou, C. Yang, Z. Shafiq, A. Liu, The rich and the poor: A markov decision process approach to optimizing taxi driver revenue efficiency, in: Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM '16, ACM, 2016, pp. 2329–2334.

[29] X. Yu, S. Gao, X. Hu, H. Park, A markov decision process approach to vacant taxi routing with e-hailing, Transportation Research Part B: Methodological 121 (2019) 114–134.
[30] X. Zhou, H. Rong, C. Yang, Q. Zhang, A. V. Khezerlou, H. Zheng, Z. Shafiq, A. X. Liu, Optimizing taxi driver profit efficiency: A spatial network-based markov decision process approach, IEEE Transactions on Big Data 6 (2020) 145–158.
[31] Z. Shou, X. Di, J. Ye, H. Zhu, R. Hampshire, Where to find next passengers on e-hailing platforms? - a markov decision process approach, arXiv preprint arXiv:1905.09906 (2020).
[32] C.-M. Tseng, S. C.-K. Chau, X. Liu, Improving viability of electric taxis by taxi service strategy optimization: A big data study of new york city, IEEE Transactions on Intelligent Transportation Systems 20 (2019) 817–829.
[33] N. Garg, S. Ranu, Route recommendations for idle taxi drivers: Find me the shortest route to a customer!, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1425–1434.
[34] Y. Gao, D. Jiang, Y. Xu, Optimize taxi driving strategies based on reinforcement learning, International Journal of Geographical Information Science 32 (2018) 1677–1696.
[35] M. Han, P. Senellart, S. Bressan, H. Wu, Routing an autonomous taxi with reinforcement learning, in: Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM '16, ACM, 2016, pp. 2421–2424.
[36] C. Yan, H. Zhu, N. Korolko, D. Woodard, Dynamic pricing and matching in ride-hailing platforms, Naval Research Logistics (NRL) 67 (2020) 705–724.
[37] H. A. Chaudhari, J. W. Byers, E. Terzi, Putting data in the driver's seat: Optimizing earnings for on-demand ride-hailing, in: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, ACM, 2018, pp. 90–98.
[38] S. Guo, Q. Shen, Z. Liu, C. Chen, C. Chen, J. Wang, Z. Li, K. Xu, Seeking based on dynamic prices: Higher earnings and better strategies in ride-on-demand services, IEEE Transactions on Intelligent Transportation Systems 24 (2023) 5527–5542.
[39] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data?, in: Proceedings of the 36th Conference on Neural Information Processing Systems, NeurIPS '22, 2022, pp. 507–520.
[40] R. Shwartz-Ziv, A. Armon, Tabular data: Deep learning is not all you need, Information Fusion 81 (2022) 84–90.
[41] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, 2016, pp. 785–794.
[42] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in: Proceedings of the 31st Conference on Neural Information Processing Systems, NIPS '17, 2017, pp. 1–9.