A Queueing-Theoretic Framework for Vehicle Dispatching in Dynamic Car-Hailing

Peng Cheng
East China Normal University
Shanghai, China
pcheng@sei.ecnu.edu.cn

Jiabao Jin
East China Normal University
Shanghai, China
10175101146@stu.ecnu.edu.cn

Lei Chen
The Hong Kong University of Science and Technology
Hong Kong, China
leichen@cse.ust.hk

Xuemin Lin
The University of New South Wales
Sydney, Australia
East China Normal University
Shanghai, China
lxue@cse.unsw.edu.au

Libin Zheng
Sun Yat-sen University
Zhuhai, China
zhenglb6@mail.sysu.edu.cn

ABSTRACT

With the rapid development of smart mobile devices, the car-hailing platforms (e.g., Uber or Lyft) have attracted much attention from both the academia and the industry. In this paper, we consider an important dynamic car-hailing problem, namely *maximum revenue vehicle dispatching* (MRVD), in which rider requests dynamically arrive and drivers need to serve as many riders as possible such that the entire revenue of the platform is maximized. We prove that the MRVD problem is NP-hard and intractable. In addition, the dynamic car-hailing platforms have no information of the future riders, which makes the problem even harder. To handle the MRVD problem, we propose a queueing-based vehicle dispatching framework, which first uses existing machine learning algorithms to predict the future vehicle demand of each region, then estimates the idle time periods of drivers through a queueing model for each region. With the information of the predicted vehicle demands and estimated idle time periods of drivers, we propose two batch-based vehicle dispatching algorithms to efficiently assign suitable drivers to riders such that the expected overall revenue of the platform is maximized during each batch processing. Through extensive experiments, we demonstrate the efficiency and effectiveness of our proposed approaches over both real and synthetic datasets. In summary, our methods can achieve 5% ~ 10% increase on overall revenue without sacrifice on running speed compared with existing solutions.

PVLDB Reference Format:
Peng Cheng, Jiabao Jin, Lei Chen, Xuemin Lin, and Libin Zheng. A Queueing-Theoretic Framework for
Vehicle Dispatching in Dynamic Car-Hailing. PVLDB, 14(1): XXX-XXX, 2021.
doi:XX.XX/XXX.XX

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

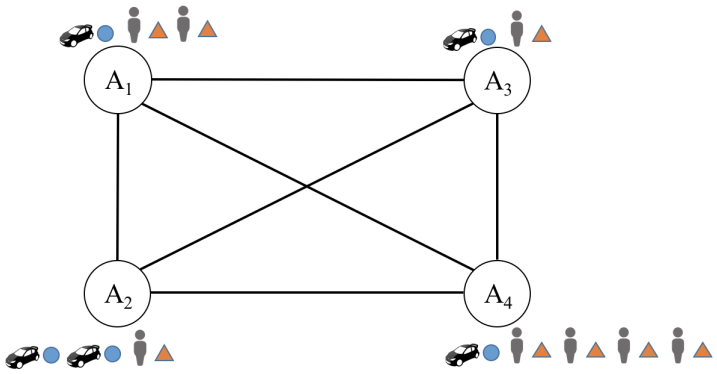


Figure 1: A multi-area vehicle dispatching example.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/haidaoxiaofei/queueing-car-hailing.

1 INTRODUCTION

Recently, with the popularity of the smart devices and high quality of the wireless networks, people can easily access network and communicate with online services. With the convenient car-hailing platforms (e.g., Uber [5] and DiDi Chuxing [1]), drivers can share their vehicles to riders to obtain monetary benefits and alleviate the pressure of public transportation. One of the crucial issues in the platforms is to efficiently dispatch vehicles to suitable riders. Although the platforms have become huge recently, during peak hours (e.g., 8 am) in some high demand areas (e.g., residential areas), riders need to wait for up to several hours before being served. To mitigate the shortage of vehicles in particular time and areas and improve the efficiency of the platforms, we investigate a queueing-theoretic framework in this paper.

We illustrate the general idea of our framework in the following motivation example.

Example 1. *Consider a scenario of taxi dispatching in Figure 1, where each one of the four inter connected areas $A_1 \sim A_4$ maintains a queue of riders and taxis. The numbers of icons of taxis and riders*

near each area reflect the ratio between them in the corresponding area. For example, in area A_1, the number of available taxis is only half of the number of waiting riders. Riders only want to wait for a limited time (e.g., 5 minutes), otherwise they will switch to other public transportation systems (e.g., the bus system). Usually, taxis can easily pickup riders within the same area (e.g., moving several hundreds meters). However, if one taxi is assigned with a rider in a different area, the taxi may need to move several kilometers to pickup the rider. In addition, after serving its current rider, one taxi will usually wander around the destination area to pickup a new rider.

When the number of taxis is less than the demanding riders, the platform needs to smartly select riders to serve, such that the total revenue of the platform can be improved. In the example in Figure 1, the platform should give higher priorities to the riders whose destination is within area A_1 or A_4, where taxis are scarce. On the contrast, the platform should give lower priorities to the riders whose destination is within area A_2, where taxis are abundant.

Motivated by the example above, in this paper, we propose a novel vehicle dispatching framework, which aims to take riders' destinations into consideration to alleviate the shortage of taxis in particular areas such that the overall revenue of the platform can be maximized.

Existing works in spatial matching or taxi dispatching only consider the pickup locations of riders and try to minimize the travel distance of taxis to pick riders [20, 23], which causes some taxis need to wait for a long time period before picking up new convenient riders after finishing their last orders leading to the low efficiency of the platform (i.e., imbalanced demand-and-supply in some regions). In our previous poster paper [7], to the best of our knowledge, we are the first to introduce the general idea of the queueing theoretic framework to balance the demand-and-supply of taxi dispatching during a relative long time period to maximize the overall revenue of the platform. However, in [7] we just explained the general queueing theoretic framework without detailed algorithms, analyses, and experimental studies, which will be introduced in this work.

To improve the overall revenue of the platform during a relatively long time period, we propose a *batch-based queueing-theoretic vehicle dispatching* framework in this paper. Specifically, we partition the whole space into regions and maintain a queue of waiting riders and drivers for each region. Once there are available drivers, the most-priority rider in the queue of waiting riders will be served. In addition, riders may *quit/renege* from the platform if she is not served for a long period. Usually, a driver, after delivering her current rider, will continue to serve the next rider around the destination of the current rider. Thus, during serving the riders, the distribution of drivers will be changed. To serve as many riders as possible, intuitively, the platform should *match* the distributions of riders and drivers (i.e., one region with more riders should have more drivers) through associating higher priorities to the riders whose destinations are in regions lacking of drivers, then drivers can serve riders quickly before they quit/renege. We first propose models to estimate the Poisson distributions of riders and drivers. Then, we utilize the queueing theory to analyzing the idle time interval for each driver after finishing his/her assigned rider. Finally, we propose two vehicle/driver dispatching algorithms to maximize the overall revenue of the platform in each batch processing. Note that, we maximize the overall revenue of the platform through improving the efficiency of the entire platform to serve more riders without increasing the charges to riders or decreasing the payment to drivers. In fact, the payment to drivers is usually a portion of the overall revenue of the platform. Thus, the more the overall revenue of the platform is, the more the payment to drivers is. In conclusion, our solution will benefit riders, drivers and the platform at the same time.

To summarize, we make the following contributions in the paper:

- We propose a batch-based queueing theoretic framework for vehicle dispatching in Section 3.
- We estimate the idle interval time of drivers in Section 4.
- We propose two vehicle dispatching algorithms for each batch processing in Section 5.
- We have conducted extensive experiments on real and synthetic data sets, to show the efficiency and effectiveness of our queueing-theoretic framework in Section 6.

In addition, the remaining sections of the paper are arranged as follows. We review and compare previous studies on queueing theory and vehicle dispatching in Section 7 and conclude the work in Section 8.

2 PROBLEM DEFINITION

In this section, we present the formal definition of the vehicle dispatching problem, where a system will assign drivers to riders to delivery them to their destinations.

In this paper, we use a graph $G = \langle V, E \rangle$ to represent a road network, where V is a set of vertices and E is a set of edges. Each edge $(u, v) \in E$ $(u, v \in V)$ is associated with a weight $cost(u, v)$ indicating the travel cost from vertex u to vertex v. Here the travel cost could be the travel time or the travel distance. When we know the travel speed of vehicles, we can convert one to another. In the rest of this paper, we will not differentiate between them and use *travel cost* consistently.

To better manage the riders and drivers, we assume the entire space is divided into a set of n regions/grids $A = \{a_1, a_2, ..., a_n\}$.

2.1 Riders and Drivers

Definition 1. (Impatient Rider) Let r_i be an impatient rider, who submit his/her order o_i to the platform at timestamp t_i, and is associated with a source location s_i, a destination location e_i and a pickup deadline τ_i.

In particular, a rider r_i comes to the platform to call for a driver to deliver him/her from his/her current location s_i to his/her destination location e_i. The request is sent to the platform at timestamp t_i. As the rider is impatient, if the platform cannot assign an available driver to pick up him/her within τ_i time after t_i, he/she will quit/renege from the platform and maybe switch to other platforms or use other transportation systems. Usually, rider r_i will not rejoin the platform immediately after he/she is delivered to his/her destination. Thus, in this paper, we assume that each rider is unique and only lives for the lifetime of his/her ride in the platform. In addition, if a rider r_i is delivered to his/her destination, the platform will charge him/her for $\alpha \cdot cost(s_i, e_i)$, where α is the travel fee rate of the platform.

Definition 2. (Driver) Let d_j be a driver, who is located at position $l_j(t)$ at timestamp t. Her status is either *busy* (i.e., on delivering any riders) or *available* (i.e., free to be assigned to a rider).

When a new driver d_j join the platform, he/she is considered to be available to serve riders. Once a rider r_i is assigned to a driver d_j, the driver will move to the source location s_i to send the rider to his/her destination location e_i. During that period, driver d_j is considered *busy*. After driver d_j finishes his/her current task, driver d_j will become available again. For region a_k at timestamp t, we denote the set of available drivers as $D_k(t)$ and the number of them as $|D_k(t)|$.

2.2 The Maximum Revenue Vehicle Dispatching Problem

Before present the formal definition of the maximum revenue vehicle dispatching problem, we first define the valid rider-and-driver dispatching pair.

Definition 3. (Valid Rider-and-Driver Dispatching Pair) Let $\langle r_i, d_j \rangle$ be a valid rider-and-driver dispatching pair, where driver d_j can arrive at the pickup location s_i of rider r_i before the pickup deadline τ_i and driver d_j is in available status when he/she is picking up rider r_i.

Now we give the formal definition of the maximum revenue vehicle dispatching problem as follows:

Definition 4. (Maximum Revenue Vehicle Dispatching Problem, MRVD) For a given time period $\mathbb{T}$, a set of impatient riders $R_{\mathbb{T}}$ and a set of drivers $D_{\mathbb{T}}$, the maximum revenue vehicle dispatching problem is to select a set, $I_{\mathbb{T}}$, of *valid rider-and-driver dispatching pairs* such that the overall revenue of the platform is maximized, which is:

$$\max \sum_{\langle r_i, d_j \rangle \in I_{\mathbb{T}}} \alpha \cdot cost(s_i, e_i), \tag{1}$$

where α is the travel fee rate of the platform.

Intuitively, to maximize the overall revenue the platform should serve as many long travel distance riders as possible as shown in Equation 1. However, the platform has no control on riders (i.e., the platform cannot schedule the arrivals and enlarge the waiting deadlines of riders), and can only affect the behaviors of drivers (i.e., dispatching drivers to pickup different riders). Thus, we have the reduction in the end of this section to show practical rules to dispatching drivers to maximize the overall revenue of the platform.

2.3 Hardness of MRVD

We prove MRVD is NP-hard through a reduction from the 0-1 Knapsack problem [26], which is a well know NP-hard problem.

Theorem 2.1. *(Hardness of MRVD) The problem of maximum revenue vehicle dispatching (MRVD) is NP-hard.*

Proof. We prove the theorem by a reduction from the 0-1 Knapsack problem [26]. A 0-1 Knapsack problem can be described as follows: given a set of m items numbered from 1 up to m, each with a weight w_i and a value v_i, along with a knapsack that has a maximum weight capacity W, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity W.

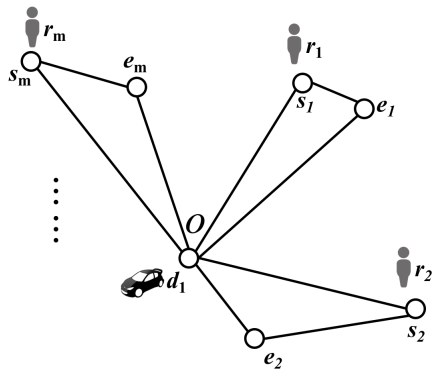


Figure 2: Illustration of the MRVD Instance.

For a given knapsack problem, we can transform it to an instance of MRVD shown in Figure 2 as follows: we give only one driver d_j with lifetime of length $T_j = W$ located at a node O at the very beginning. In addition, we give a set of m riders joining the platform at timestamp 0 and locating at different source locations, and the pickup deadlines of riders are all set to W. For each rider r_i, we set $cost(O, s_i) + cost(s_i, e_i) + cost(e_i, O) = w_i$ and $\alpha \cdot cost(s_i, e_i) = v_i$. To satisfy the triangle inequality, we pick a parameter α such that for any rider r_i we have $\frac{v_i}{\alpha} < \frac{1}{2} w_i$. Then, for this MRVD instance, we want to arrange a schedule for the given driver such that his/her overall revenue is maximized.

Since no routes between the branches of riders and the traveling cost to serve any rider and come back to location O is equal to w_i, to serve a rider r_i, the vehicle needs to go to node s_i then goes back to node O before serving the other riders, which leads to a traveling time cost of w_i and increases the revenue value with v_i. As the deadlines of all the requests are set to W, the vehicle can finish ride requests before W. Thus, to maximize the overall revenue satisfying the pickup deadlines of riders is same to maximize the total value of the items in the knapsack problem under the constraint of the capacity of the given knapsack.

Given this mapping it is easy to show that the knapsack problem instance can be solved if and only if the transformed MRVD problem can be solved.

This way, we reduce the 0-1 Knapsack problem to the MRVD problem. Since the 0-1 Knapsack problem is known to be NP-hard [26], MRVD is also NP-hard, which completes our proof. □

2.4 Reduction of MRVD

Let T_j be the lifetime of driver d_j from the time he/she joins to the time he/she exits the platform. During T_j, the status of driver d_j keeps switching between *available* and *busy*. We notice that when driver d_j is in *busy* status, he/she contributes to the overall revenue of the platform. Then we can rewrite the objective function of MRVD as below:

Table 1: Symbols and Descriptions.

Symbol	Description
a_k	a region/grid
r_i	an impatient rider
o_i	the order of the impatient rider r_i
t_i	the timestamp when r_i posts her ride request
s_i	the source location of rider r_i
e_i	the destination location of rider r_i
τ_i	the pickup deadline of rider r_i
d_j	a driver
$l_j(t)$	the position of driver d_j at timestamp t
$D_k(t)$	a set of available drivers in region a_k at time t

$$\begin{aligned} &\max \sum_{\langle r_i, d_j \rangle \in I_{\mathbb{T}}} \alpha \cdot cost(s_i, e_i) \\ \Rightarrow &\max \sum_{d_j \in D_{\mathbb{T}}} \sum_{r_i \in R_j} \alpha \cdot cost(s_i, e_i) \\ \Rightarrow &\max \alpha \sum_{d_j \in D_{\mathbb{T}}} \sum_{r_i \in R_j} cost(s_i, e_i) \end{aligned} \tag{2}$$

where $D_{\mathbb{T}}$ is the set of drivers on the platform during the given time period $\mathbb{T}$, and R_j is the set of riders that are served by driver d_j. According to Equation 2, the platform should maximize the length of the total busy time of each driver to maximize its overall revenue. Since the lifetime T_j of each driver d_j is fixed, to maximize his/her total *busy* time, $\sum_{r_i \in R_j} cost(s_i, e_i)$, is equivalent to minimize his/her total idle time, $T_j - \sum_{r_i \in R_j} cost(s_i, e_i)$. Then, the objective of MRVD can be rewritten as follows:

$$\begin{aligned} &\max \alpha \sum_{d_j \in D_{\mathbb{T}}} \sum_{r_i \in R_j} cost(s_i, e_i) \\ \Rightarrow &\min \sum_{d_j \in D_{\mathbb{T}}} \Big(T_j - \sum_{r_i \in R_j} cost(s_i, e_i) \Big) \\ \Rightarrow &\min \sum_{d_j \in D_{\mathbb{T}}} \sum_{i=0}^{|R_j|} \psi_{ij} \end{aligned} \tag{3}$$

where ψ_{ij} is the idle time of driver d_j after delivering rider r_i. Here ψ_{0j} indicates the idle time of driver d_j before picking up his/her first rider.

According to Equation 3, to maximize the overall revenue, the platform intuitively should reduce the number of served riders (e.g., $|R_j|$) and the time interval (e.g., ψ_{ij}) between any two consecutive riders for each driver. However, it may be confused that reducing the number of served riders for each driver seem to contradict the goal of maximizing the overall revenue. In fact, when the lifetime of a driver d_j is fixed, fewer served riders for driver d_j means that the average travel cost of the served riders is higher. Then, we can have two practical and controllable rules for the platform to maximize its overall revenue during a given time period $\mathbb{T}$: **a)** ***associating higher priorities to the riders whose travel costs are high*****; b)** ***reducing the length of the idle time between serving any two consecutive riders for each driver.***

In the rest of this paper, we propose a queueing-theoretic framework, taking into consideration of the travel cost and the idle time length after each ride request, to maximize the overall revenue of the platform during a given time period.

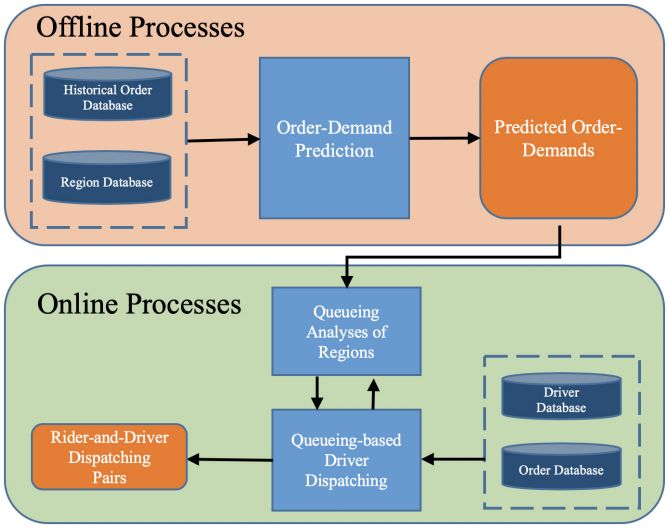


Figure 3: Illustration of the Framework Work Flow.

3 OVERVIEW OF QUEUEING-BASED VEHICLE DISPATCHING FRAMEWORK

In this section, we introduce an overview of our queueing-based vehicle dispatching framework. In general, the framework includes three parts: vehicle demand-supply prediction, queueing analysis of each region and queueing-based vehicle dispatching algorithms. From the other perspective, our framework contains offline prediction processes and online analysing and dispatching processes as shown in Figure 3. In the offline processes, the platform predicts the number of orders for each region in each time periods based on the historical order records. In the online processes, the platform utilizes the predicted order demands and supplies to assign available drivers to orders with a goal to maximizing the overall revenue of the platform subjected to the deadline constraint of orders. In addition, the driver dispatching process and the queueing analyses process can affect each other, thus we interactively assign orders to drivers and update the results of queueing analyses of regions.

We first briefly introduce the major parts of our queueing-based vehicle dispatching framework, then propose a batch-based vehicle dispatching algorithm to handle the orders from riders.

3.1 Major Parts of Queueing-Based Vehicle Dispatching Framework

Our queueing-based vehicle dispatching framework includes three major parts: offline vehicle demand-supply prediction, region queueing analysis and queueing-based vehicle dispatching algorithm.

3.1.1 Offline Vehicle Demand-Supply Prediction. In our framework, we predict the order demand of each region for given time periods. For the rejoined drivers, we can estimate their availability based on their assignments and travel costs. In practice, it is hard to predict the accurate location and timestamp of a particular rider since the uncertain behaviors of a single user. To utilize the distribution of

riders, we predict the number of riders for a given region (i.e., a spatial range of area, such as square regions or hexagon regions) in a given time period (i.e., next 30 minutes). Existing work can be applied offline to predict the demand of riders in given regions and time periods, such as demand-supply prediction of traffic [11, 19], and spatial-temporal data prediction [14, 28]. In this paper, we tested the representative state-of-the-art prediction algorithms (e.g., Historical Average method, Gradient Based Regression Tree method [16] and DeepST with Graph Convolutional Network [18, 27]) on the real-world taxi demand-supply dataset and select the most effective one, DeepST with Graph Convolutional Network (DeepST-GC), for our offline demand-supply prediction process. Due to space limitation, we put the detailed comparison of the spatial temporal prediction models in Appendix A of our technical report.

3.1.2 Region Queueing Analysis. The available drivers in a region a_x in a time period $\mathbb{T}$ come from the rejoined active drivers. With the predicted numbers of orders for the region a_x in a given time period $\mathbb{T}$ and the schedules of active drivers, we can know the demand and supply of drivers for the region a_x in time period $\mathbb{T}$. We estimate the waiting times (idle time intervals) for vehicles from finishing last order to receiving next order in Section 4. According to the analysis in Section 2, shorter idle time intervals are better. Thus, the estimated waiting times of vehicles can be used to guide the order dispatching process to achieve a high overall revenue.

3.1.3 Queueing-Based Vehicle Dispatching. The platform needs to dispatch drivers to serve most "valuable" riders with high priorities. According the analyses in Section 2, orders having high travel costs and ending in "hot regions" (i.e., regions with many future orders) can contribute more to the platform, which should be associated with high priorities. In addition, since drivers usually prefer to serve riders close to their locations after they finish the last orders, the platform's selection on serving orders will affect the vehicle supply in future, which in turn will affect the queueing analyses of the related regions. In Example 1, if the platform dispatch a driver to serve a rider having a destination of region A_1, the driver supply in region A_1 will increase slightly after finishing the order. We propose efficient and effective algorithms in Section 5 to dispatch available drivers to riders with an optimization goal of maximizing the overall revenue of the platform subjected to the deadline constraint of orders.

3.2 The Batch-based Vehicle Dispatching Algorithm

To handle the online processes of vehicle dispatching, we propose a batch-based processing framework to iteratively assign drivers to riders every Δ seconds. Note that, in real applications (e.g., DiDi Chuxing [1]), Δ is set very small (e.g., several seconds) such that the customers cannot notice the delay of the batch processing. To solve the assignment problem in each batch, we propose two heuristic algorithms to greedily maximize the revenue summation of the platform for the current scheduling time period $[\bar{t}, \bar{t} + t_c]$, where $\bar{t}$ indicates the current timestamp and t_c is the length of the current scheduling time period.

As shown in Algorithm 1, we iteratively assigns drivers to riders for multiple batches with a time interval Δ between every two

Algorithm 1: Batch-based Vehicle Dispatching Algorithm

Input: The overall time period $\mathbb{T}$
Output: A set of rider-and-driver dispatching pairs within the time period $\mathbb{T}$

1 **while** *current time* $\bar{t}$ *is in* $\mathbb{T}$ **do**
2 **foreach** $a_k \in A$ **do**
3 retrieve the waiting riders in region a_k to R_k
4 retrieve the available drivers in region a_k to D_k
5 predict the number of upcoming riders in region a_k during $[\bar{t}, \bar{t} + t_c]$ as $|\hat{R}_k|$
6 count the number of upcoming rejoined drivers in region a_k during $[\bar{t}, \bar{t} + t_c]$ as $|\hat{D}_k|$
7 use *task-priority greedy* or *local search* approach to obtain a set of rider-and-driver pairs $I_{\bar{t}}$
8 **foreach** $\langle r_i, d_j \rangle \in I_{\bar{t}}$ **do**
9 inform driver d_j to pick rider r_i
10 wait till $\bar{t} + \Delta$

successive batches. Specifically, for a batch starting at timestamp $\bar{t}$, we first retrieve a set, R_k, of waiting riders and a set, D_k, of available drivers for each region a_k (lines 3-4). Here, waiting riders R_k include the riders that are not assigned with any drivers during the last batch and the newly coming riders after the last batch in region a_k. Moreover, available drivers D_k includes the drivers that are not assigned with any riders in the last batch, and the drivers that have finished the previous assigned tasks then rejoin the platform in region a_k. To estimate the arrival rates of riders and serving rates drivers for the current scheduling time period $[\bar{t}, \bar{t} + t_c]$, we predict the number, $|\hat{R}_k|$, of upcoming riders and estimate the number, $|\hat{D}_k|$, of rejoin drivers in region a_k (lines 5-6). We will introduce the used prediction method in Section A. Then, we use our proposed heuristic vehicle dispatching algorithms to achieve a set $I_{\bar{t}}$of rider-and-driver dispatching pairs to greedily maximize the revenue summation of the platform for the current scheduling time period $[\bar{t}, \bar{t} + t_c]$ (line 7). For every rider-and-driver dispatching pair $\langle r_i, d_j \rangle$ in $I_{\bar{t}}$, we inform the driver d_j to pick up rider r_i (lines 8-9). Finally, we wait until the time comes to the next batch $\bar{t} + \Delta$ (line 10).

In the following sections, we will first introduce the queueing analyses of regions in Section 4, then propose our queueing based vehicle dispatching algorithms in Section 5.

4 QUEUEING ANALYSES OF REGIONS

In this section, we analyze the waiting riders for each single region through a queueing model. In queueing theory, customers join queue in an arrival (or "birth") rate λ, then the platform will serve the customers in a service ("death") rate μ. We first introduce the queue configuration, then estimate the idle time for a driver after he/she finishes his/her current order.

4.1 Queue Configuration of a Single Region

In this paper, the platform can be considered as a server to match available drivers and waiting riders in each region. The riders come to the platform and wait for drivers to pick them. However, the riders are impatient and will leave the platform if they are not

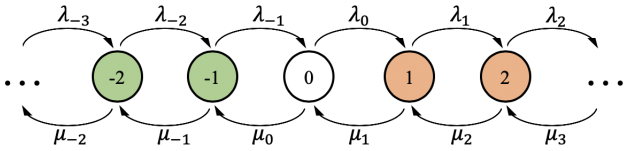


Figure 4: Birth-death chain for the queue of a region.

served before their deadlines. The available drivers come from the rejoined active drivers, who are the ones continuing to work on the platform after finishing their assigned orders.

Similar to the previous assumption in the related work [6], we assume the arrival rate of riders (in number per minute) follow the Poisson distribution with rates λ in a region a during a short time period with length t_c (e.g., a half hour). In addition, we also model the arrivals of rejoined active drivers follow a Poisson distribution with a rate of μ in a region a during a short time period with length t_c. Note that, although the arrival rate of riders and drivers may change during different time periods in a day (e.g., 8 to 9 A.M. and 8 to 9 P.M.), to facilitate the analysis of the queueing situation in a short time period (e.g., a half hour), we model the arrival rates of riders and drivers as stable rates.

When the riders are more than the drivers in a region, the platform will select a subset of riders with "higher priorities" to serve first. Usually a driver will rejoin the platform in the same region of the destination of her/his last served rider. Thus, the drivers will rejoin the platform in the regions where the destinations of the selected high-priority riders are. In our queue model, the priority of a rider is determined in line with his/her travel cost and the demand-supply situation in his/her destination region. According to the analysis in Section 2.4, to improve the overall revenue the platform prefers to give higher priorities to the riders who have higher travel costs and are going to "hot" regions.

Figure 4 illustrates the birth-death chain of the queueing model for each region, where each circle indicates a state and the numbers in the circles are the number of waiting riders in the corresponding states. For example, the state of **2** indicates there are 2 waiting riders in the region. Each link represents the transfer event from the tail state to the head state along its direction, where the value close to the link indicates the transfer rate. For example, the link with value λ_0 directing from state **0** to state **1** indicates the transfer rate of the region states from **0** to **1** is λ_0. Since drivers may also congest in a region if the arrival rate of drivers is higher than that of riders (i.e., $\mu > \lambda$), to uniformly represent the queueing situation of a region, we utilize the state of **-n** ($-n < 0$) to indicate that there are n congested drivers in the region.

Another issue to change the number of waiting riders is that the impatient riders may quit from the platform if they are not served after a time period, which is called *reneging* in queueing theory. As defined in the existing work [21], we can define a state related reneging function $\pi(n)$ as the reneging rate of riders for the state $\mathbf{n}$ ($n > 0$) of the birth-death chain in Figure 4. Suggested in [21], a good practice for the reneging function $\pi(n)$ is to define it as $e^{\beta n/\mu}$, where β is a parameter determined based on the historical reneging records in the corresponding region. Then, the death/service rate μ_n of the state $\mathbf{n}$ can be adjusted as follows:

$$\mu_n = \begin{cases} \mu, & n \leq 0 \\ \mu + \pi(n), & n > 0 \end{cases} \tag{4}$$

For the birth/arrival rate λ_n of state $\mathbf{n}$, we define it as $\lambda_n = \lambda$, since drivers do not renege in our queue model.

4.2 Expected Idle Time Interval of Drivers

In this section, we analyze the expected idle time interval of a driver d_j. Let region a be the destination region of the last rider r_i of d_j. After finishing delivering r_i, driver d_j will join the queue of region a. Let p_n denote the probability that the queue of region a is in state $\mathbf{n}$.

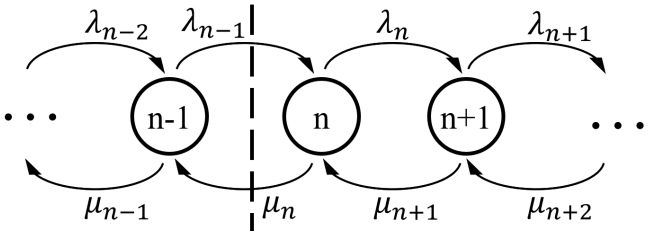


Figure 5: Flow balance between states.

We here briefly introduce a method to analyze the recursive relation between p_n and p_{n-1} mentioned in [21]. As shown in Figure 5, the mean flows across the dash line must be equal in steady state for the queue. In other words, for a relatively long period (e.g., 30 minutes), the rate of transitions ($\lambda_{n-1}p_{n-1}$) from state **n-1** to state **n** must equal the rate of transitions ($\mu_n p_n$) from state **n** to state **n-1**, which is as follows:

$$\mu_n p_n = \lambda_{n-1} p_{n-1} \tag{5}$$

By iteratively applying Equation (5), we can derive:

$$p_n = \begin{cases} p_0 \cdot (\frac{\mu}{\lambda})^{-n}, & n < 0 \\ p_0 \cdot \prod_{i=1}^{n} \frac{\lambda}{\mu + \pi(i)}, & n > 0 \end{cases} \tag{6}$$

We next analyze the expected idle time in different conditions: 1) more riders arrive; 2) more drivers rejoin; 3) balanced riders and drivers, since they have different properties.

4.2.1 More Riders Arrive ($\lambda > \mu$). When $\lambda > \mu$, the summation of probabilities of states $\{\mathbf{n}\}$, $n < 0$, can be calculated through a summation of geometric sequence, which is:

$$\sum_{i=-1}^{-\infty} p_i = p_0 \sum_{i=1}^{\infty} (\frac{\mu}{\lambda})^i = \frac{\mu}{\lambda - \mu} p_0 \tag{7}$$

In addition, we have the fact that the summation of the probabilities $\{p_n\}, n \in [-\infty, \infty]$, must be 1.

$$\sum_{i=-1}^{-\infty} p_i + p_0 + \sum_{i=1}^{\infty} p_i = 1 \tag{8}$$

Putting Equations (6) and (7) into Equation 8, we have:

$$p_0 \Big(\frac{\mu}{\lambda - \mu} + 1 + \sum_{n=1}^{\infty} \prod_{i=1}^{n} \frac{\lambda}{\mu + \pi(i)} \Big) = 1$$

Then, we have

$$p_0 = \Big(\frac{\lambda}{\lambda-\mu} + \sum_{n=1}^{\infty}\prod_{i=1}^{n}\frac{\lambda}{\mu+\pi(i)}\Big)^{-1} \tag{9}$$

Drivers are dispatched in a first come, first served order. Let $T(n)$ be the expected idle time interval of an arrival driver d_j when the queue is in state $\mathbf{n}$. When there are waiting riders ($n > 0$), the idle time interval of d_j is just the processing time of the platform to arrange a new rider, which can be ignored. When there is no waiting riders ($n \leq 0$), the driver needs to wait until the next $(|n|+1)$th rider appears in the region, which needs $\frac{|n|+1}{\lambda}$ time on average. Finally, we can estimate the expected idle time, $ET(\lambda, \mu)$, of a driver after join a queue of region having the arrival rate λ of riders and the arrival rate μ of drivers as follows:

$$\begin{aligned} ET(\lambda,\mu) &= \sum_{i=0}^{-\infty}\frac{|n|+1}{\lambda}\cdot p_i + \sum_{i=1}^{\infty} 0\cdot p_i \\ &= \frac{p_0}{\lambda}\sum_{i=0}^{\infty}(i+1)\big(\frac{\mu}{\lambda}\big)^i \\ &= \frac{\lambda p_0}{(\lambda-\mu)^2} \end{aligned} \tag{10}$$

4.2.2 More Drivers Rejoin ($\lambda < \mu$). We notice that when $\lambda < \mu$, the queue will congest more and more drivers when time elapses, which will harm the efficiency of the platform much. The platform will avoid that the rate of drivers μ become larger than the rate of riders λ for each region. However, when there are indeed more drivers rejoining, we still can estimate the expected idle time.

Let K be the number of available drivers during the current scheduling time period with length t_c. Then, the queue of the region can at most congest with K drivers. Let $\theta = \frac{\mu}{\lambda}$. We can calculate the summation of probabilities of states $\{\mathbf{n}\}$, $-K \leq n < 0$, with the equation as follows:

$$\sum_{i=-1}^{-K} p_i = p_0\sum_{i=1}^{K}\big(\frac{\mu}{\lambda}\big)^i = \frac{\theta^{K+1}-\theta}{\theta-1}p_0 \tag{11}$$

Then, we can update Equation 9 when $\lambda < \mu$ as follows:

$$p_0 = \Big(\frac{\theta^{K+1}-1}{\theta-1} + \sum_{n=1}^{\infty}\prod_{i=1}^{n}\frac{\lambda}{\mu+\pi(i)}\Big)^{-1} \tag{12}$$

In addition, when the expected number of rejoined drivers during the current scheduling time period with length t_c is K and $\lambda < \mu$, we can estimate the expected idle time $ET(\lambda, \mu)$ as follows:

$$ET(\lambda,\mu) = \frac{p_0}{\lambda}\frac{(K+1)\theta^{K+2}-(K+2)\theta^{K+1}+1}{(\theta-1)^2}. \tag{13}$$

4.2.3 Balanced Riders and Drivers ($\lambda = \mu$). When $\lambda = \mu$, we can update Equation 11 as follows:

$$\sum_{i=-1}^{-K} p_i = p_0\sum_{i=1}^{K}\big(\frac{\mu}{\lambda}\big)^i = Kp_0 \tag{14}$$

Then, we have

Algorithm 2: Idle Ratio Oriented Greedy Algorithm

Input: A set of Regions A, current timestamp $\bar{t}$
Output: A set of rider-and-driver dispatching pairs $I_{\bar{t}}$

```
I_t̄ ← {∅}
I_v ← {∅}
foreach a_k ∈ A do
    retrieve a set I_k of valid rider-and-driver dispatching
      pairs from R_k and D_k
    I_v ← I_v ∪ I_k
    estimate the arrival rate λ_(k) of riders and arrival rate
      μ_(k) of rejoined drivers in region a_k during [t̄, t̄ + t_c]
sort dispatching pairs in I_v based on their idle ratio
while I_v is not empty do
    select the rider-and-driver pair ⟨r_i, d_j⟩ having the
      smallest idle ratio from I_v
    add ⟨r_i, d_j⟩ to I_t̄
    update μ_(k) of the destination region a_k of r_i
    remove ⟨r_i, .⟩ and ⟨., d_j⟩ from I_v
return I_t̄
```

$$p_0 = \Big(K+1+\sum_{n=1}^{\infty}\prod_{i=1}^{n}\frac{\lambda}{\mu+\pi(i)}\Big)^{-1} \tag{15}$$

Next, we can estimate the expected idle time $ET(\lambda, \mu)$ when $\lambda = \mu$ as follows:

$$ET(\lambda,\mu) = p_0\frac{(K+1)(K+2)}{2\lambda} \tag{16}$$

5 QUEUEING-BASED VEHICLE DISPATCHING ALGORITHMS

5.1 The Idle Ratio Oriented Greedy Approach

We first propose an *idle ratio oriented greedy* approach to solve each batch process in line 7 of Algorithm 1 with a goal to maximize the revenue summation of the platform during the current scheduling time period $[\bar{t}, \bar{t}+t_c]$, where $\bar{t}$ is the current timestamp and t_c is the length of the current scheduling time window. We first define the idle ratio of driver d_j to server rider r_i, whose destination e_i is in region a_k, as follows:

$$IR(r_i, d_j) = \frac{ET(\lambda_{(k)}, \mu_{(k)})}{cost(s_i, e_i) + ET(\lambda_{(k)}, \mu_{(k)})}, \tag{17}$$

where $ET(\lambda_{(k)}, \mu_{(k)})$ is the expected idle time of driver d_j when he/she rejoins the platform at region a_k, and $cost(s_i, e_i)$ is the travel cost (travel time) on serving rider r_i. Recall that, in Section 2.4, we have two guiding rules for the platform to maximize its overall revenue after analyzing the MRVD problem: **a) *associating higher priorities to the riders whose travel costs are higher*; b) *reducing the length of the idle time between serving any two consecutive riders for each driver.*** We notice that when the travel cost $cost(s_i, e_i)$ increases, $IR(r_i, d_j)$ will decrease; when the expected idle time $ET(\lambda_{(k)}, \mu_{(k)})$ decreases, $IR(r_i, d_j)$ will also decrease. As a result, we only need to greedily select the rider-and-driver dispatching pairs with low idle ratio (as defined in Equation 17), then

we can follow the above mentioned two guiding rules to maximize the overall revenue of the platform. Based on the observation, we propose an idle ratio oriented greedy approach as shown in Algorithm 2, which greedily select the rider-and-driver dispatching pair having the smallest current idle ratio value.

Specifically, we first initialize the selected rider-and-driver pairs $I_{\bar{t}}$ and the valid rider-and-driver pairs I_v with empty sets (lines 1-2). Then, for each region a_k, we put the valid rider-and-driver pairs I_k between the waiting riders R_k and available drivers D_k in the region into I_v (lines 4-5) and estimate the arrival rates, $\lambda_{(k)}$ and $\mu_{(k)}$, of new riders and rejoined drivers during the current scheduling period $[\bar{t}, \bar{t} + t_c]$ as follows:

$$\lambda_{(k)} = \begin{cases} \frac{|\hat{R}_k|}{t_c}, & |R_k| \leq |D_k| \\ \frac{|\hat{R}_k| + |R_k| - |D_k|}{t_c}, & |R_k| > |D_k| \end{cases} \tag{18}$$

$$\mu_{(k)} = \begin{cases} \frac{|\hat{D}_k| + |D_k| - |R_k|}{t_c}, & |R_k| \leq |D_k| \\ \frac{|\hat{D}_k|}{t_c}, & |R_k| > |D_k| \end{cases} \tag{19}$$

where $|\hat{R}_k|$ and $|\hat{D}_k|$ are the numbers of predicted riders and future rejoined drivers in region a_k during $[\bar{t}, \bar{t} + t_c]$. Next, after retrieving all the valid pairs, we sort them based on their idle ratios calculated with Equation 17 (line 7). Note that, the expected idle time $ET(\lambda_{(k)}, \mu_{(k)})$ is determined by the arrival rates, $\lambda_{(k)}$ and $\mu_{(k)}$, of new riders and rejoined drivers in the destination region a_k, thus we only need to estimate that for each region but not for each rider-and-driver pair individually. In each iteration of the while-loop (lines 8-12), we select the rider-and-driver pair $\langle r_i, d_j \rangle$ having the smallest idle ratio and remove related pairs, $\langle r_i, . \rangle$ and $\langle ., d_j \rangle$, of rider r_i and driver d_j from I_v (since each driver only can serve one rider at one time). The selected pair $\langle r_i, d_j \rangle$ is added in $I_{\bar{t}}$ (line 10) and all the selected pairs $I_{\bar{t}}$ will be finally returned (line 13).

Complexity Analysis. Let the number of total waiting riders be m, the number of total available drivers be n and the number of total regions be x. Assume riders and drivers be evenly distributed in x regions and x is much smaller than m and n. In lines 3-6 of algorithm 2, retrieving all the valid rider-and-driver pairs needs $O(\frac{mn}{x})$. To sort the valid pairs in I_v needs $O(\frac{mn}{x} \log_2(\frac{mn}{x}))$ (line 7). In each iteration of the while-loop (lines 8 - 12 of Algorithm 2), selecting the pair having the smallest idle ratio from sorted I_v needs $O(1)$ (lines 9-10); updating $\mu_{(k)}$ and the idle ratio of average $\frac{mn}{x^2}$ related pairs needs $O(\frac{mn}{x^2})$ (line 11); removing the related valid pairs $\langle r_i, . \rangle$ and $\langle ., d_j \rangle$ from I_v needs $O(\max(\frac{n}{x}, \frac{m}{x}))$ (line 12). Since in each iteration, at least one rider and one driver will be matched, thus there will be at most $\min(m, n)$ iterations. Then the complexity of the while-loop is $O(\frac{\min(m,n)mn}{x^2})$. Thus, the complexity of Algorithm 2 is $O(\max(\frac{mn}{x} \log_2(\frac{mn}{x}), \frac{\min(m,n)mn}{x^2}))$. If we consider x as a constant number, and m is linearly related to n, the complexity can be considered as $O(n^3)$.

5.2 The Local Search Algorithm

In the idle ratio oriented greedy approach, we greedily select the pair having the "current" smallest idle ratio. However, the arrival rate μ_k of rejoined drivers in region a_k will change after selecting some riders whose destinations are in a_k. Thus, the idle ratios of early selected rider-and-driver pairs may increase later on. To overcome the shortcoming of the idle ratio oriented greedy approach, in this section, we will propose a local search algorithm to improve the results, which keeps searching for rider-and-driver pairs $\langle r'_i, d_j \rangle$ with a smaller idle ratio for driver d_j and update the assigned rider of d_j to r'_i until no such pairs can be found.

Algorithm 3: Local Search Algorithm

Input: A set of Regions A, current timestamp $\bar{t}$
Output: A set of rider-and-driver dispatching pairs $I_{\bar{t}}$

```
1  Obtain a set, I_t̄, of pairs with Algorithm 2
2  do
3      FLAG ← False
4      foreach ⟨r_i, d_j⟩ ∈ I_t̄ do
5          foreach r'_i ∈ R_j do
6              if IR(r'_i, d_j) < IR(r_i, d_j) then
7                  update ⟨r_i, d_j⟩ to ⟨r'_i, d_j⟩
8                  FLAG ← True
9  while FLAG is True
10 return I_t̄
```

Specifically, in Algorithm 3, we first obtain a set, $I_{\bar{t}}$, of rider-and-driver pairs for current timestamp $\bar{t}$ through Algorithm 2 (note that, we can also obtain $I_{\bar{t}}$ through any other algorithms). Then, in each iteration, we check whether the rider of a pair $\langle r_i, d_j \rangle \in I_{\bar{t}}$ can be replaced by any other valid rider $r'_i \in R_j$ for d_j, where R_j is the valid riders to d_j. If no replacement happens, we will return the updated set, $I_{\bar{t}}$, of the selected rider-and-driver pairs (line 10).

We prove our local search algorithm can converge through the below lemma.

Lemma 5.1. *Algorithm 3 can converge.*

Proof. Assume Algorithm 3 cannot converge. Then, there is at least one driver d who keeps switching between two riders r_u and r_v. When d selects r_u, we have $IR(r_u, d) < IR(r_v, d)$; otherwise, we have $IR(r_u, d) < IR(r_v, d)$. We denote the regions where r_u and r_v end as a_u and a_v, respectively.

Since the travel costs of r_u and r_v do not change, the reason that $IR(r_u, d)$ and $IR(r_v, d)$ change is that $ET(\lambda_u, \mu_u)$ and $ET(\lambda_u, \mu_u)$ change. Specifically, according to the definition of $IR(r, d)$ in Equation 17, $IR(r, d)$ is positively correlated with $ET(\lambda_u, \mu_u)$ (e.g., when $ET(\lambda, \mu)$ increases, $IR(r, d)$ will also increase). In addition, when more drivers rejoin in region a_u, the expected waiting time, $ET(\lambda_u, \mu_u)$, will increase.

If driver d switches from r_u to r_v in some iteration ζ, there must be at least one other driver d' switching from his/her valid rider r'_v to r'_u (i.e., $IR(r'_u, d') < IR(r'_v, d')$). Thus, we have $IR(r_u, d) > IR(r_v, d)$. After d switches to r_v, the number of rejoined drivers in region a_u will decrease, and $IR(r'_u, d')$ will also decrease. As a result, d' will not switch back to r'_v and d will not switch back to r_u, which is contradicted with the assumption that d keeps switching between r_u and r_v. Thus Algorithm 3 can converge. □

Complexity Analysis. Let the number of total waiting riders be m, the number of total available drivers be n and the number of total regions be x. Assume riders and drivers be evenly distributed

in x regions and x is much smaller than m and n. The number of total valid rider-and-driver pairs will be $O(\frac{mn}{x})$. The number, $|R_j|$, of valid riders for driver d_j will be $O(\frac{n}{x})$. Then each iteration of the while-loop needs $O(\frac{mn^2}{x^2})$. Let L_{max} be the maximum iteration numbers, then the complexity of Algorithm 3 will be $O(L_{max}\frac{mn^2}{x^2})$. If we consider x and L_{max} as constant numbers, and m is linearly related to n, the complexity can be considered as $O(n^3)$.

6 EXPERIMENTAL STUDY

In this section, we show the efficiency and effectiveness of our queueing-theoretic framework with different vehicle dispatching algorithms embedded through experimental study on both synthetic and real datasets.

6.1 Data Sets

We use both real and synthetic data to test our framework. Specifically, for the real data set, we use the taxi trip data sets in NYC [2].

New York Taxi Trip Dataset. New York Taxi and Limousine Commission (TLC) Taxi Trip Data [2] is a dataset recording the information of taxi trips in New York, USA. The records are collected and provided to the NYC Taxi and Limousine Commission technology under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP [4]). Trip records can be categories as three types: yellow taxi, green taxi and FHV (For Hire Vehicle). However, due to the privacy issues, only the locations of yellow taxi can be access in the dataset long time ago. In addition, the number of FHV and green taxi records is much smaller than that of yellow taxi. Thus, we only use the taxi trip records of yellow taxis in our experiments. Each trip record includes the taxi's pick-up and drop-off taxi-zones and GPS locations and timestamps, the number of passengers and the total travel cost. In the dataset, there are 262 taxi zones from zone 2 to zone 263. In our experiment, we use taxi trip data records from July 2012 to December 2012 as training data set and from 1st to 3rd of January 2013 as the test data set.

6.2 Experimental Configurations

For the experiments on the real data set, we use the pickup location and timestamp of a taxi trip record to initialize the source location s_i and the posting timestamp t_i of a ride order r_i. Then the dropoff location of the taxi trip record is used to set the destination e_i of the ride order. For the pickup deadline τ_i of rider r_i, we configure it by adding a random noise $\tau' \in [1, 10]$ and a base pickup waiting time τ (configured with the setting in Table 2) to the posting timestamp t_i (e.g., $\tau_i = t_i + \tau' + \tau$). To initialize the origin location $l_j(0)$ of driver d_j at the beginning timestamp 0, we first randomly generate zone a_k for the driver following the distribution of the number of orders over the whole 262 zones. Then uniformly generate the location $l_j(0)$ in zone a_k.

In our experiment, we run the batch process every time period Δ. To estimate the arrival rate of new riders and rejoined drivers, we look up a time window of length t_c with the "current" timestamp $\bar{t}$ as the beginning point.

Table 2: Experimental Settings.

Parameters	Values
the number, n, of drivers	800, 1K, 2K, **3K**, 4K
base pickup waiting time, τ (seconds)	60, **120**, 180, 240, 300
the length of batch interval, Δ (seconds)	**3**, 5, 10, 20, 30
the length of time window, t_c (minutes)	**5**, 10, 15, 20, 40, 60, 80, 100

6.3 Approaches and Measurements

We conduct experiments to evaluate the effectiveness and efficiency of our queueing-theoretic vehicle dispatching framework with two batch processing vehicle dispatching algorithms, namely *idle ratio oriented greedy* (IRG) and local search (LS), in terms of the total revenue and the average batch running time. Note that, we set the parameter α as 1, such the total revenue is equal to the total serving time (e.g., the summation of the travel cost of all the served ride orders).

Specifically, for IRG (or LS) we can further have two different combinations: IRG-P and IRG-R (or LS-P and LS-R), which use the predicted taxi demand and the real taxi demand, respectively. In addition, we also compare our approaches with three baseline methods: (1) *long trip greedy* (LTG), which greedily assigns orders with the highest revenue to available taxis; (2) *nearest trip greedy* (NEAR), which greedily assigns the nearest order to each available taxi; (3) *random* (RAND), which randomly assigns orders to available taxis. We also compare our methods with the state-of-the-art solution, POLAR [24], on car-hailing problem, which utilizes the predicted number of orders and drivers to conduct an offline bipartite matching first, then uses the offline result as a blueprint to guide the only task matching. In addition, we also report the upper bound (UPPER) by summing up the revenue of the most expensive orders that can be served by idle drivers ignoring their distances in each batch.

Table 2 shows the settings of our experiments, where the default values of parameters are in bold font. In each set of experiments, we vary one of the parameters and keep other parameters in their default values. For each experiment, we report the average total revenue and the average batch processing time of our tested approaches for a whole day (from 00:00:00 to 23:59:59). All our experiments were run on an Intel Xeon X5675 CPU @3.07 GHZ with 32 GB RAM in Java. The code of our queueing-theoretic vehicle dispatching framework and prediction methods can be accessed in our github project [3].

6.4 Experimental Results of the Estimated Idle Time

In this section, we evaluate the accuracy of our queueing theoretic model on estimating the idle time of the drivers after finishing their assigned tasks. To show the results, we vary the number of drivers from 1K to 10K and keep the other parameters in their default values as shown in Table 2. We report the mean average error (MAE), relative root mean square error (RMSE) and real root mean square error (Real RMSE) of our estimated waiting time periods of drivers compared with their real waiting time periods in Table 3.

From the results, we find that our queueing theoretic model can achieve good estimated idle time periods of the drivers after finishing their assigned tasks. When the number of drivers increases from 1K to 10K, the MAE, RMSE and real RMSE first decrease then increase. The reason is that our default batch interval is 3

Table 3: Results of the Estimated Idle Time

#Drivers	MAE (s)	RMSE (%)	Real RMSE (s)
1K	2.48	4.91	12.14
2K	2.11	4.97	9.33
3K	1.83	4.72	5.02
4K	1.97	5.11	7.04
6K	2.17	5.42	9.23
8K	2.51	5.92	12.14
10K	5.33	6.72	41.92

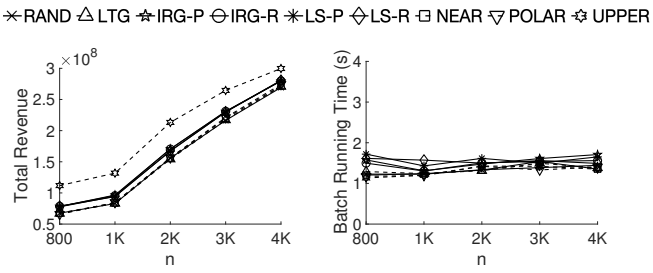
seconds, when the number of drivers is 1K, the drivers can almost immediately receive new task after they finish their assigned tasks. However, due to the batch process, they need to wait until next batch process, which in fact leads to the major difference between the estimated waiting time periods and the real ones. When the number of drivers increases from 1K to 4K, more and more drivers needs to wait for a while to receive a new task after they finish their last tasks. Then the estimation errors causing by the batch processes become tiny. When the number of drivers continues increasing from 4K to 10K, the idle time of drivers also increases obviously. The MAE and real RMSE of the results of our queueing theoretic model also increases obviously, however the relative RMSE only increases 1.5%, which shows that our estimation model is accurate.

6.5 Experimental Results of Vehicle Dispatching Approaches

In this section, we show the effects of the number, n, of drivers, the base pickup waiting time τ, the length, Δ, of batch interval, and the length, t_c, of time window to estimate the arrival rates of riders and rejoined drivers.

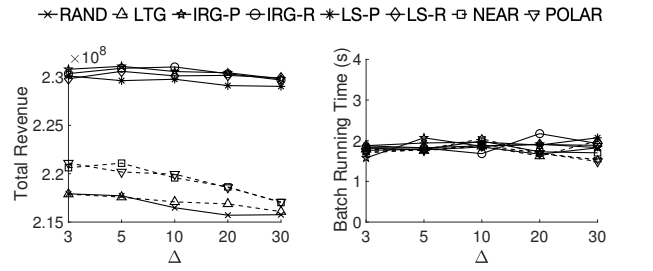
Effect of the Number, n, of Drivers. Figure 6 illustrates the experimental results on varying the number of drivers from 1K to 10K, where other parameters are in their default values. In Figure 6(a), when the number of drivers increases from 800 to 4K, all the tested approaches can achieve results with increasing total revenue. The reason is that when more drivers are available, more riders can be served before their pickup deadlines. When the number of drivers is 800, our IRG and LS approaches can achieve higher total revenue than RAND, LTG, NEAR and POLAR. The difference between the results of our IRG and LS are small. When the number of drivers increases, the advantage of our IRG and LS in terms of the total revenue become narrow. We also notice that when the number of drivers reaches 4K, all the tested approaches can achieve results with total revenue close to the upper bound. The reason is that when there are 4K drivers, almost all the riders can be served as long as he/she joins the platform. The vehicle dispatching algorithms have no difference in term of the total revenue, when drivers are sufficient. To clearly show the differences between the total revenues of our tested approaches, we will not plot out the results of UPPER as they are always same with the results in Figure 6(a).

In Figure 6(b), when the number of drivers increases, the batch running time of all the tested approaches also increases slightly, which is because in each batch there are more drivers to process requiring more time to process. We can see that all the tested approaches can finish each batch processing within 2 seconds, which



(a) Total Revenue (b) Batch Running Time

Figure 6: Effects of Number of Drivers n.



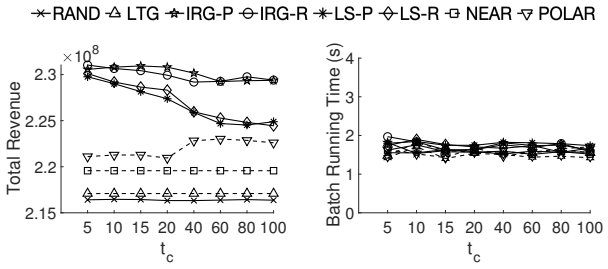
(a) Total Revenue (b) Batch Running Time

Figure 7: Effects of Batch Length Δ.

is unnoticeable to the users and acceptable for the batch processes with 3-second intervals.

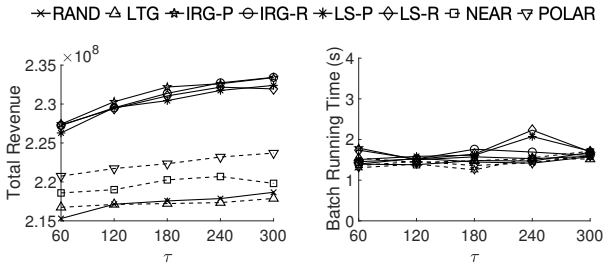
Effect of the Length, Δ, of Batch Interval. Figure 7 shows the experimental results on varying the length, Δ, of the batch interval from 3 to 30 seconds, while other parameters are set to their default values. As shown in Figure 7(a), when the length, Δ, of batch interval increases from 3 to 30 seconds, the total revenues of the results achieved by the tested approaches decrease slightly. The reason is that when the length of the batch interval increases, more riders may be unserved before their pickup deadlines between two consecutive batches. In other words, when Δ increases, the probability of a rider becomes time out will increase during the batch intervals, when the platform does not respond to any riders or drivers. Another reason is that when drivers become available, they also need to wait for the next batch to be assigned with new riders, which also leads to the bad effect on the total revenue. Thus, in real applications, Δ should not be too large. In addition, we notice that our IRG-P and LS-P can achieve higher total revenues than RAND, LTG, NEAR and POLAR. We find that when we use the ground truth of the taxi demand for our IRG-R and LS-R algorithms, they can achieve higher total revenues than IRG-P and LS-P, which shows the importance of the accuracy of the taxi demand methods. In other words, for the real applications, a more accurate prediction model can bring increases on the total revenue of the system.

In Figure 7(b), the batch running time of the tested approaches slightly increases, since the number of riders and drivers for each batch will increase when the length, Δ, of batch interval increases. The differences of the batch running time of the tested approaches are small.



(a) Total Revenue (b) Batch Running Time

Figure 8: Effects of Time Window t_c.



(a) Total Revenue (b) Batch Running Time

Figure 9: Effects of Base Waiting Time τ.

Effect of the Length, t_c, of Time Window. Figure 8 presents the experimental results on varying the length t_c of time window on estimating the arrival rate of new riders and rejoined drivers. In Figure 8(a), the total revenue achieved by IRG and LS will decrease when t_c becomes larger than 20 minutes. The reason is that most taxi trips in NYC taxi trip dataset have a travel time of less than 20 minutes [10]. The effect of future rejoined drivers in more than 20 minutes later is almost neglectable for our IRG and LS algorithms. However, when t_c becomes 40 minutes, POLAR can perform better than it in the experiment of t_c smaller than 20 minutes. Our IRG and LS algorithms can still outperform POLAR in terms of total revenue when t_c is larger than 40 minutes. Thus, in real platform, t_c should not be too large. Since RAND and LTG do not consider the demand and supply of the taxis in future, the length, t_c, of time window has no effect on them. In Figure 8(b), t_c has no clear effect on the running time of our tested approaches.

Effect of the Base Waiting Time τ. Figure 9 illustrates the effect of the waiting time τ of riders by varying τ from 60 to 300 seconds while keeping other parameters in their default values. In Figure 9(a), when the waiting time of riders τ increases, the total revenue of the results achieved by our tested approaches also increases. The reason is that when riders can wait for longer time, the probability that they can be served by some drivers will increase, which is consistent with human intuition. With the help of ground truth of the taxi demand (more accurate than our predicted demand), LS-R can achieve slightly higher total revenue than LS-P. IRG, LS and their variants can all surpass RAND, LTG, NEAR and POLAR. In Figure 9(b), the batch running time of tested approaches increases slightly when the waiting time of riders increases. The reason is that when riders can wait for longer time and the number of drivers does not change, the number of riders in each batch will also increase, which leads to the processing time of the batch becomes longer.

In summary, LS and IRG can all beat RAND, LTG, NEAR and POLAR in terms of total revenue. The accuracy of taxi demand prediction method can affect the final results on the total revenue. Thus, taxi demand prediction models with higher accuracy are more valuable for the platform. Our framework is efficient. In all the experiments, the running time of each batch for all the tested approaches is less than 2 seconds, which is affordable for the platform to perform a batch process with 3 seconds for each batch interval.

7 RELATED WORK

Recently, with the rapid development of online car-hailing platforms, the daily active users of the platform grows up in a tremendously speed, which has drawn attention from both academia and industry.

Our MRVD problem is related to task assignment in spatial crowdsourcing [8, 9, 17, 24], which assign a set of workers to the locations of tasks to conduct subject to various constraints and optimization goals. However, in our MRVD problem, each order has a pickup location and a destination, while each task in spatial crowdsourcing usually has only one required location. In [17], based on the publishing models, the authors classified the spatial crowdsourcing in two modes: worker selected task (WST) mode [15] and server assigned tasks (SAT) mdoe [8, 9, 24]. In WST mode, workers select tasks by themselves. In SAT mode, the server/system has the control on assign tasks to workers base on its objectives. In SAT mode, there are two processing styles: online task assignment mode [24] and batch-based task assignment mode [8, 9, 17]. Recently, researchers start to utilize the prediction models to predict the future distributions of workers and tasks to improve the overall performance in a relatively long time period (e.g., 1 day). For instance, researchers build an offline blueprint based on the predicted distributions of workers and tasks, then use it to guide the online task assignment to maximize the total number of assigned tasks [24]. Our MRVD targets on maximizing the total revenue of the platform, which cannot apply existing solutions directly. Thus, we develop our queueing theoretic framework, which uses queueing theory to estimate the idle time of drivers based on the predicted number of orders and drivers in each region.

Our MRVD problem is also related to dial-a-ride problem (DARP), which assume a fleet of vehicles located at a common depot, and schedules should be made to accommodate m rider requests based on their pick-up and drop-off time constraint. Existing works on DARP have mainly focused on static offline DARP, where the constraints are known beforehand. The general DARP is NP-hard and intractable, unless its scale is not big (e.g., hundreds of vehicles and riders) [12]. [13] uses a heuristic method called tabu search to find the neighbourhood solution from current solution, to avoid finding cycle result and local optimum, they forbid the recent visited answers and use some diversification mechanism.

With the emergence of ridesharing business, many riders and drivers prefer choosing the ridesharing service, as it is cheaper than non-share car request with limited time delay, and it improves the transportation efficiency in the congested urban cities. [10] designs a algorithm to dispatch the similar rider to the same car with a

goal of maximizing the total utility, which includes the rider related utility, vehicle-related utility and trajectory-related utility. In [29], the authors propose a packing-based approach, which first packs the riders together then assigns groups of riders to vehicles. To solve the scheduling problem for a vehicle with a set of assigned riders, authors in [25] propose a linear time complex method. However, ridesharing mainly focuses on scheduling and solving conflicts of route-sharable riders to vehicles, which is different in our MRVD setting.

In addition, traffic prediction is also a critical technology in urban city transportation scenario. With accurate prediction, we can foresee the future and make plan to fulfill the long time revenue. There are many models which focusing on predicting the number of orders in the next time slot by integrating temporal and spatial information. [27] proposes a Deep ST model which combines the geographical and historical traffic data together, to decrease the difference between estimated traffic flow number and actual count. With the powerful deep convolutional neural network and rich daily meta data (e.g., holiday and weather), they get the state of art prediction results.

8 CONCLUSION

In this paper, we study the problem of maximum revenue vehicle dispatching problem (MRVD), in which rider requests dynamically arrive and drivers need to serve as many riders as possible such that the entire revenue of the platform is maximized. We prove that the MRVD problem is NP-hard and intractable. Through analyses, we find to maximize the total revenue, we need to give higher priorities to ride orders with long travel cost and less idle time. We propose a queueing-theoretic framework, which predicts the taxi demand (rider orders) offline and schedule the drivers to regions where the idle time of them will be small. Our framework dispatching drivers to riders in a batch-based processing for every Δ seconds. To handle the batch vehicle dispatching problem, we propose two heuristic algorithms, namely idle ratio oriented greedy (IRG) and local search (LS). Through experiments on the real and synthetic data sets, we show the effectiveness and efficiency of our queueing-theoretic vehicle dispatching framework.

REFERENCES

[1] 2021. [Online] DiDi Chuxing. https://www.didichuxing.com.
[2] 2021. [Online] NYC Taxi & Limousine Commission Trip Record Data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.
[3] 2021. [Online] Source Code of Queueing-Theoretic Vehicle Dispatching Framework. https://www.AvailableSoon.com.
[4] 2021. [Online] Taxicab Passenger Enhancements Project. http://www.nyc.gov/html/tlc/html/industry/taxicab_serv_enh.shtml.
[5] 2021. [Online] Uber. https://www.uber.com.
[6] Siddhartha Banerjee, Ramesh Johari, and Carlos Riquelme. 2016. Dynamic pricing in ridesharing platforms. *ACM SIGecom Exchanges* 15, 1 (2016), 65–70.
[7] Peng Cheng, Chao Feng, Lei Chen, and Zheng Wang. 2019. A queueing-theoretic framework for vehicle dispatching in dynamic car-hailing. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1622–1625.
[8] Peng Cheng, Xiang Lian, Lei Chen, Jinsong Han, and Jizhong Zhao. 2016. Task assignment on multi-skill oriented spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2201–2215.
[9] Peng Cheng, Xiang Lian, Zhao Chen, Rui Fu, Lei Chen, Jinsong Han, and Jizhong Zhao. 2015. Reliable diversity-based spatial crowdsourcing by moving workers. *Proceedings of the VLDB Endowment* 8, 10 (2015), 1022–1033.
[10] Peng Cheng, Hao Xin, and Lei Chen. 2017. Utility-aware ridesharing on road networks. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1197–1210.
[11] Jing Chu, Kun Qian, Xu Wang, Lina Yao, Fu Xiao, Jianbo Li, Xin Miao, and Zheng Yang. 2018. Passenger Demand Prediction with Cellular Footprints. In *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 1–9.
[12] Jean-François Cordeau. 2006. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research* 54, 3 (2006), 573–586.
[13] Jean-François Cordeau and Gilbert Laporte. 2003. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological* 37, 6 (2003), 579–594.
[14] Noel Cressie and Christopher K Wikle. 2015. *Statistics for spatio-temporal data*. John Wiley & Sons.
[15] Dingxiong Deng, Cyrus Shahabi, and Ugur Demiryurek. 2013. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems*. 324–333.
[16] Jerome H Friedman. 2002. Stochastic gradient boosting. *Computational Statistics & Data Analysis* 38, 4 (2002), 367–378.
[17] Leyla Kazemi and Cyrus Shahabi. 2012. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 20th international conference on advances in geographic information systems*. ACM, 189–198.
[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[19] Yexin Li, Yu Zheng, Huichu Zhang, and Lei Chen. 2015. Traffic prediction in a bike-sharing system. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 33.
[20] Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. 2010. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering* 7, 3 (2010), 607–616.
[21] John F Shortle, James M Thompson, Donald Gross, and Carl M Harris. 2018. *Fundamentals of queueing theory*. Vol. 399. John Wiley & Sons.
[22] N Kipf Thomas and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint. *arXiv preprint arXiv:1609.02907* 103 (2016).
[23] Yongxin Tong, Jieying She, Bolin Ding, Lei Chen, Tianyu Wo, and Ke Xu. 2016. Online minimum matching in real-time spatial data: experiments and analysis. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1053–1064.
[24] Yongxin Tong, Libin Wang, Zhou Zimu, Bolin Ding, Lei Chen, Jieping Ye, and Ke Xu. 2017. Flexible online task assignment in real-time spatial data. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1334–1345.
[25] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. 2018. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1633–1646.
[26] Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.
[27] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. (2017), 1655–1661.
[28] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, and Xiuwen Yi. 2016. DNN-based prediction model for spatio-temporal data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 92.
[29] Libin Zheng, Lei Chen, and Jieping Ye. 2018. Order dispatch in price-aware ridesharing. *Proceedings of the VLDB Endowment* 11, 8 (2018), 853–865.

A METHODS AND RESULTS OF OFFLINE DEMAND PREDICTION

We first introduce the methods we used to predict the demand of taxi (the number of riders) for each zone in each time slot (with length of 30 minutes). With the predicted numbers of riders for 262 regions, we can estimate the arrival rate of new riders in a given zone at a given time period.

To predict the future order number, we use different kinds of information, such as the order counts from previous time slots, and other meta data (e.g., time of day, day of week and city weather). We use the modified model of DeepST [27], which uses previous order numbers from three different time scales: *closeness*, *period*, *trend*. Here, *closeness* means the previous N time slots; *period* indicates the same time in previous N days; *trend* refers to the same time in previous N weeks. Meanwhile, it also uses other features (e.g., weather information) to make a good prediction by use Convolutional Neural Network [18].

Since New York taxi zones are not regular rectangle grids, we cannot use Convolutional Neural Network directly to extract the adjacent information. To solve this issue, we replace the conv layer as Graph Convolutional layer, which comes from Graph Convolutional Network [22]. We name our modified model as DeepST-GC (DeepST with Graph Convolutional Network). Next, we introduce the basic component of our model: Graph Convolution Layer.

We represent the whole New York City as a graph $G = \{V, E\}$, where each taxi zone is represented as a vertex and their connectivities is regarded as edges. All nodes' features can be stored in a matrix X whose dimension is $[N, F]$, N means the number of taxi-zone, F indicates the number of features of each taxi-zone. Connectives matrix, $\tilde{A}$, is added with an identity matrix, I, to form the adjacency matrix. The adjacency matrix helps to pass the features of nodes to their related nodes. Before we use the adjacency matrix, we normalize the matrix such that all the summations of each row is equal to 1. Let D be the diagonal node degree matrix, then the normalized adjacency matrix would be:

$$A = D^{-\frac{1}{2}}(\tilde{A} + I)D^{-\frac{1}{2}}$$

The whole graph convolution computation is the formulation:

$$X^{t+1} = f(X^t, A) = \sigma(A^T X^t W^t),$$

where X^t and W^t are the input and weights of the tth layer of the neural network.

We use separated model to process three different features of categories, then add all the output together to get the final result. Meanwhile, we use day of week and time of day embedding feature from a lookup feature table.

We use the first five months to train the demand count prediction model, and use the last month to test the performance of our dispatching algorithm. During prediction, we use 30 minutes as a time slot. Table 4 lists the statistic information of the taxi trip record data set.

To evaluate the accuracy of our DeepST-GC, we report the RMSE loss as the evaluation metric. To show the effectiveness of the model, we compare our model with the following prediction models:

Table 4: The statistics of the taxi trip dataset

	Yellow Taxi
Days of Train Records	91
Days of Evaluation Records	20
Days of Test Records	10
max Records Count Per Slot	853
min Records Count Per Slot	0
max Records Count Per Zone	12017
min Records Count Per Zone	0

- HA. Historical Average calculates the mean of the order records in the previous 15 time slots as the next cat order count.
- LR. Linear Regression model collects the order records in the previous 15 time slots to predict the next car order count.
- GBRT. Gradient Based Regression Tree [16] model collects the order records in the previous 15 time slots and uses non-parametric regression to predict the next car order count.

The evaluation results are shown on the table 5, we can see that the DeepST GC model performs best. Thus, we only use DeepST-GC model in the following experiments.

Table 5: Results of the Demand Prediction Methods

	RMSE (%)	Real RMSE (s)
DeepST-GC	2.30	15.03
HA	7.46	48.21
LR	3.40	21.66
GBRT	2.74	17.67