# InBEDE: Integrating Contextual Bandit with TD Learning for Joint Pricing and Dispatch of Ride-Hailing Platforms

Haipeng Chen[1,2,*], Yan Jiao[2,**], Zhiwei (Tony) Qin[2,**], Xiaocheng Tang[2,**],
Hao Li[2,**], Bo An[1,*], Hongtu Zhu[2,**], and Jieping Ye[2,**]

[1]Nanyang Technological University
[2]AI Labs, Didi Chuxing
[*]{chen0939,boan}@ntu.edu.sg
[**]{yanjiao, qinzhiwei, xiaochengtang, karenlihao_i, zhuhongtu, yejieping}@didiglobal.com

***Abstract*—For both the traditional street-hailing taxi industry and the recently emerged on-line ride-hailing, it has been a major challenge to improve the ride-hailing marketplace efficiency due to spatio-temporal imbalance between the supply and demand, among other factors. Despite the numerous approaches to improve marketplace efficiency using pricing and dispatch strategies, they usually optimize pricing or dispatch separately. In this paper, we show that these two processes are in fact intrinsically interrelated. Motivated by this observation, we make an attempt to simultaneously optimize pricing and dispatch strategies. However, such a joint optimization is extremely challenging due to the inherent huge scale and lack of a uniform model of the problem. To handle the high complexity brought by the new problem, we propose InBEDE (Integrating contextual Bandit with tEmporal DiffErence learning), a learning framework where pricing strategies are learned via a contextual bandit algorithm, and the dispatch strategies are optimized with the help of temporal difference learning. The two learning components proceed in a mutual bootstrapping manner, in the sense that the policy evaluations of the two components are inter-dependent. Evaluated with real-world datasets of two Chinese cities from Didi Chuxing, an online ride-hailing platform, we show that the market efficiency of the ride-hailing platform can be significantly improved using InBEDE.**

***Index Terms*—Ride-hailing platform, Joint pricing and dispatch, Reinforcement learning**

## I. Introduction

In the taxi industry, the problem of spatio-temporally imbalanced taxi supply and trip demand has been a major obstacle of market efficiency for decades. When demand is higher than supply (e.g., during peak hours or in central business areas), passengers suffer from long response time of trip requests, while in the opposite case, drivers bare losses from idling around without fulfilling passenger requests. The rapid revolution of the taxi industry from street hailing to on-line ride-hailing platforms such as Uber, Lyft and Didi Chuxing has provided the technological levers to alleviate the supply-demand imbalance. However, tackling this problem still presents a significant challenge.

Numerous approaches have been proposed to mitigate the problem and thus improve ride-hailing market efficiency. One thread of approaches study the question that, when a passenger inputs the desired origin and destination from the ride-hailing APP (a status called "*request*"), what price should be quoted to that request. This thread of approaches utilize dynamic pricing as a leverage to coordinate supply and demand [1]–[4]. Another thread of methods focus on the question that, when a passenger finally sends out the request (a status referred to as "*order*"), which driver will be dispatched to the order. These methods evolve from greedy (local optimal) approaches which find the nearest driver to a passenger [5], [6], to globally optimal dispatch strategies with short-term [7] or long-term goals [8], [9]. One major limitation of these approaches is that they only optimize pricing and dispatch strategies separately.

We will show that, taking the long-term effect into consideration, pricing strategies and dispatch strategies are not independent, but are in fact interrelated with each other. To this end, we propose a joint dynamic pricing and dispatch framework for the on-line ride-hailing platform. One recent approach [10] studies joint pricing and dispatch in ride-hailing platforms. However, it only optimizes the length of time window for collecting orders from customers, not exactly the dispatch process. To the best of our knowledge, this is the first paper in the literature that jointly optimizes pricing and dispatch in an explicit way.

Reinforcement learning (RL) [11] has been proven to be effective in large-scale sequential planning problems, where interaction with the environment is usually described as a Markov Decision Process (MDP). However, we will show that, existing RL algorithms (e.g., DQN [12], A3C [13], DDPG [14] and TRPO [15]) cannot be directly applied in our problem, since there is no uniform way of defining the "state" and "action" for the consecutive pricing and dispatch that satisfies the real-world needs of pricing and dispatch tasks. To handle this new challenge, we propose a novel learning framework, InBEDE (Integrated contextual Bandit and tEmporal DifferencE learning, pronounced "in-bee-dee")

This work was done when Haipeng Chen was an intern at Didi Chuxing.

2374-8486/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDM.2019.00016

towards the joint pricing and dispatch problem. The contextual bandit component is deployed at each request and updated *on the go*, while the TD learning component is utilized to estimate the future effect of a pricing strategy as well as a dispatch strategy, and is updated at a lower frequency, e.g., *at the end of a day*. The two key components (i.e., contextual bandit and temporal difference learning) of InBEDE are trained in a mutually bootstrapping fashion, where the policy evaluations of two components are dependent on one another.

Our key contributions are summarized as follows:

- First, we make the first attempt in the literature to provide a uniform optimization framework for joint pricing and dispatch in on-line ride-hailing platforms.
- Second, we propose a novel solution algorithm, InBEDE, to solve the formulated joint pricing and dispatch problem. InBEDE integrates the training of contextual bandit with temporal difference learning in a mutually bootstrapping manner.
- Lastly, we conduct extensive experimental evaluations of our proposed framework using real-world datasets of two cities in China, provided by a ride-hailing platform, Didi Chuxing. The results show that by using our proposed approach, the system efficiency (in terms of total driver income and customer payment) of the ride-hailing platform has been significantly improved.

The remainder of the paper is as follows. Section II discusses related work. Section III introduces an overview of pricing and dispatch in ride-hailing platforms. Section IV describes the problem formulation. The solution algorithm is presented in Section V, followed by the experimental evaluations in Section VI. We conclude the paper in Section VII.

## II. Related Work

### A. Pricing and Dispatch in the Taxi Industry

Pricing has been adopted as a leverage to coordinate travel demand and supply and improve the ride-hailing system revenue. Prior works provide analytical studies over linear and non-linear pricing [16], static and dynamic pricing [17], or joint waging (to drivers) and pricing problem [18]. Some other works address the pricing problem from an optimization point of view and formulate the pricing problem as an MDP [1], [2]. One major limitation in these works is that the platform uses over-simplified dispatch models such as a queueing model [17], [18] or that supply is always sufficient [2].

At the same time, a variety of approaches have been proposed to address the demand-supply imbalance from the perspective of order dispatch. This thread of approaches focus on designing optimal matching algorithms among orders (demand) and drivers (supply). They evolve from greedy approaches which find the nearest driver to a passenger [5], or matching drivers with passengers on a first-come-first-serve basis [6], to globally optimized dispatch strategies with either short-term [7] or long-term goals [8], [9], [19], [20]. These approaches usually neglect the pricing strategies and assume that the prices for the travel requests are fixed.

One recent approach [10] studies both pricing and dispatch in ride-hailing platforms. However, it only optimizes the length of time window for collecting orders from customers, not exactly the dispatch process. To the best of our knowledge, this is the first paper that jointly optimizes pricing and dispatch.

### B. Multi-Armed Bandit and TD Learning

Over the past few decades, there have been extensive studies on *multi-armed bandit* problems. In multi-armed bandit problems, a set of *actions* need to be sequentially selected in order to maximize the expected reward, where rewards of different actions are partially known and may become better understood by exploring the actions. Both optimal [21], [22] and approximate (while scalable) [23], [24] algorithms have been developed to solve the problems. In most algorithms (e.g., UCB1 [23]), action selection is usually performed in a way that trades-off *exploiting* the current optimal action and *exploring* potentially better actions. More recently, researchers have been interested in a more complicated *contextual bandit* problem (also referred to as *associate search* in [25]) where the reward of an action depends on the *context* of the problem. Both linear (e.g., LinUCB [26], Linear Thompson Sampling [27]) and non-linear methods (e.g., GPUCB [28], KernelUCB [29], NeuralBandit [30]) are proposed to solve this class of problems. In linear methods, the expected reward is represented as a linear function of a context-action pair, while in non-linear methods, the authors utilize either a kernelized [29] or a neural network representation [30] of the expected reward. More recently, bandit algorithms have been integrated with collaborative filtering and clustering [31], [32] in networked scenarios.

Despite its theoretical and practical advantages in solving online learning problems, traditional multi-armed bandit algorithms are usually limited to *one-shot decision* settings. On the other hand, TD learning [33] is a family of reinforcement learning methods for sequential decision making problems, where the core idea is to bootstrap learning of value function based on current estimate of the value function. Opposed to Monte Carlo methods which can update the value function only after getting a complete trajectory of training samples, TD learning has the advantage of learning in a fully online manner and thus has been widely adopted in various prominent RL approaches such as DQN [12], A3C [13], DDPG [14] and AlphaGo [34]. In the following, we will show that due to the inherent huge scale and lack of a uniform problem model, a single RL framework such as contextual bandit algorithm and TD learning cannot be directly applied in our problem.

## III. Overview of Pricing and Dispatch

Figure 1 illustrates the pricing and dispatch process of an on-line ride-hailing system. By "request", we refer to the status where a passenger has input the trip origin and destination, but has not sent out the request. Upon the emergence of a trip request, the system quotes an estimated price to it. After the price is revealed, the passenger decides whether or not to submit the trip request based on his or her willingness to pay. Once submitted, the status of the request changes to an "order", which is put in the system's order list and is ready to be matched to a vacant driver. In practice, the order dispatch proceeds in a batch manner, e.g., with a time window of a few seconds. Within this time window, a set of requests arrive, among which a subset of them turn into a list of orders, and then the system performs a matching between the orders and drivers. This process repeats throughout a day.

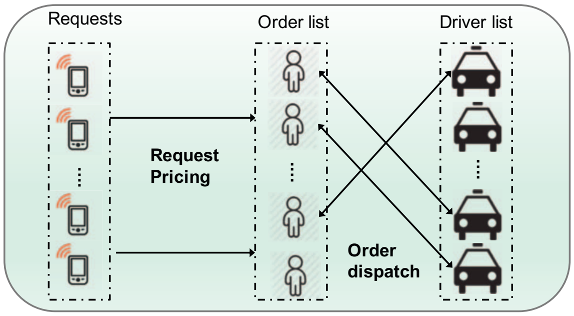


Fig. 1. Joint request pricing and order dispatch overview

A pricing strategy consists of two components, 1) a base price which is a fixed price determined by the travel distance and travel time, etc, and 2) a pricing factor which is a multiplication over the base price. Note that in our setting, the base price is an external input, and our approach only considers the pricing factors. One key idea of our approach is that, instead of focusing on the immediate effect of a pricing strategy, we also consider its future effects. Intuitively, we encourage, with a reduced price, the conversion to orders of requests from a 'cold' area to a 'hot' area.[1] Later, we will show how to quantitatively characterize this property. After a driver is assigned to the passenger and drives the passenger to a hot area, the driver is more likely to be able to fulfill another order immediately. This mitigates the supply-demand imbalance, while improving the operational efficiency of the platform. As we can see, the future effect of a request pricing strategy is reflected in the repositioning of a driver, from its original position at the current time to its destination at a future time. Thus, the future effect of repositioning the driver is highly dependent on the dispatch strategies, making it sub-optimal to optimize pricing without considering dispatching.

A dispatch strategy is usually a matching algorithm between the orders and drivers. As shown in [8], [9], [19], an optimal dispatch strategy should also consider the future effect of a matching, which assigns higher priorities to a matching with higher immediate and future potential values. Similar to the pricing of a request, the future effect of a matching is also reflected in the repositioning of a driver. Moreover, the future spatio-temporal values of the drivers are also highly dependent on the pricing strategies, in the sense that pricing strategies will affect the demand distribution of the ride-hailing system. Thus, it is sub-optimal to perform dispatch without taking into account the pricing strategies. In this paper, we introduce a novel dynamic joint request pricing and order dispatch framework to further mitigate the supply-demand imbalance, and thus to improve efficiency of the ride-hailing platform.

## IV. Joint Pricing and Dispatch

As described in Section III, there are two stages for a newly arrived travel request, namely request pricing (or equivalently, order generation) and order dispatch. In this section, we introduce a joint pricing and dispatch framework, which consists of distributed pricing and centralized dispatch.

### A. Distributed Request Pricing

In practice, whenever a passenger inputs the origin and destination of a trip request (i.e., a request emerges), the ride-hailing system needs to set a price to the request immediately (in a scale of milliseconds). As a result, it is impractical to globally optimize the pricing strategies for all the current requests due to the time needed to collect all the requests' information as well as strategy calculation. Instead, we choose to optimize the price of each request individually, thus allowing a distributed and scalable implementation.

Typically, a request $i$ is characterized by a $d$-dimensional vector of contextual features $x_i = \langle x_{ij} \rangle$, including the time $t_i$ it emerges from the on-demand ride-hailing platform, its original location $l_i$ and destination $l'_i$, the estimated base price $p_i$ (determined by the estimated trip distance and time, etc.) of the trip, the distance and estimated travel time of the trip, and historical request conversion rate of location and time, etc. Note that we do not use any customer-specific features in our framework. On top of the base price $p_i$, we impose a pricing factor $a_i \in \mathcal{A}$ to influence the probability $f(x_i, a_i)$ of the request converting into an order, which we refer to as *request conversion rate* (CR). Here $\mathcal{A}$ is the feasible space of the price factors. In practice, we consider a set of discretized price factors, e.g., $\mathcal{A} = \{0.85, 0.9, 0.95, 1, 1.05, 1.1, 1.15\}$. Intuitively, $f(x_i, a_i)$ is a non-increasing function of $a_i$, i.e., when price increases, the probability of a request converting into an order decreases, and vice versa. Given the pricing strategy $a_i$ to a request $i$, if the converted order is dispatched, we define the *immediate reward* as the expected driver income of the request:

$$r(x_i, a_i) = f(x_i, a_i)(p_i a_i - p_i \beta), \tag{1}$$

where $\beta$ is a fixed number denoting the percentage of revenue shared by the ride-hailing platform. Note that in the current setting, we characterize system efficiency as the total drive income of a day on the ride-hailing platform. It is worth mentioning that our framework can be generally adapted to other system efficiency metrics.

**Future effect.** In addition to the immediate reward, we also take into account the future effect of the current pricing strategy $a_i$ to a request $i$. When a request transforms into an order, the order dispatch system will dispatch a certain driver $j$ to the order. After the dispatch, the driver starts from the original place $l_j$, goes to the order's (which is transformed

[1] A "cold (hot)" area has less (more) demand than supply.

from the request $i$) original place $l_i$, picks up the passenger and drives the passenger to the destination $l'_i$. Consequently, this incurs the reposition of the driver $j$ from $l_j$ to $l'_i$.

### B. Global Order Dispatch Optimization

Before we discuss future effects of a request pricing strategy in detail, we first introduce the order dispatch component, to which the driver value function is a key element.

*1) Value Function and Pseudo-MDP of a Driver:* To approximate the spatio-temporal value of a driver at a certain location and time, we adapt the work of [8], [9], and define a "pseudo-MDP" for each driver. We call it a "pseudo-MPD" because the value of a driver does not only dependent on the dispatch action of the driver, but also on the pricing strategies of requests. In the pseudo-MDP, the *state* $s_j = (l_j, t_j)$ of a driver $j$ consists of the location $l_j$ and time $t_j$ of the driver. Note that the state $s_j$ of a driver is different from the contextual feature $x_i$ of a request $i$. With a bit abuse of notation, we use $i$ to denote the order that is converted from a request $i$. For a driver $j$ at state $s_j$, we restrict the set of feasible orders (for assigning drivers to) to orders that are within a certain distance (e.g., 5 kilometers) of the driver, which we denote as $\mathcal{I}_j$. The dispatch *action* of a driver is thus denoted as a binary vector $b_j = \langle b_{ji} \rangle, \forall i \in \mathcal{I}_j$. It is required that a driver can be assigned to at most one order at a time:

$$\sum_{i \in \mathcal{I}_j} b_{ji} \leq 1.$$

When a driver $j$ is assigned to an order $i \in \mathcal{I}_j$, i.e., $b_{ji} = 1$, the driver will pick up the passenger at location $l_i$, and finally go to the destination $l'_i$ of the order. In this case, the reward is $p_i a_i - p_i \beta$, where $a_i$ is the price factor. When the driver is not assigned to any order, i.e., $b_{ji} = 0, \forall i \in \mathcal{I}_j$, the driver is idle (within the next time period), and the reward is zero. For simplicity, we assume an idle driver performs a *random walk* around the original location. The random walk process is simulated with historical driver trajectory data. Therefore, the immediate reward of a driver is denoted as:

$$r(s_j, b_j) = \sum_{i \in \mathcal{I}_j} b_{ji}(p_i a_i - p_i \beta) \tag{2}$$

For *state transition*, if the driver is assigned to an order $i$, the next state is the destination of the order and the time of arrival, which is the sum of the time to pick up the passenger and the service time. If the driver is not assigned to any order, the next state is determined by the random walk process.

Different from traditional MDPs where the value of a state only depends on the action performed on the same entity (i.e., a driver), in our problem, the value of a driver also depends on the pricing strategies of the other entities (i.e., the requests). This is because the pricing strategies affect whether the requests will convert into orders, and thus the feasible order set $\mathcal{I}_j$ would be affected for each driver $j$. To take this into account, we use $\pi$ to denote the generic joint pricing and dispatch policy, and the immediate reward of a driver $j$ (Eq. (2)) can be rewritten as $r_\pi(s_j)$. Thus, the accumulated value of a driver $j$ at state $s_j = (l_j, t_j)$ is defined as:

$$V_\pi(s_j) = \sum_{s'_j = s_j}^{s^{end}} r_\pi(s'_j), \tag{3}$$

where $r_\pi(s'_j)$ is the reward of a driver under state $s'_j$ with the policy $\pi$, and $s^{end}$ is the *terminal state*. In this problem, the terminal state of a driver is defined as the state either one day is over or the driver logs off the ride-hailing platform.

*2) Order Dispatch:* The order dispatch process aims at assigning drivers to orders, so that the orders are served. In practice, order dispatch usually takes place on a discrete time basis (e.g., every few seconds). Within the current time period, a set $\mathcal{I}$ of orders (including those that are left from the last time period) are collected, and there are a set $\mathcal{J}$ of vacant drivers that are distributed over the entire city. Given a matching of a driver $j \in \mathcal{J}$ and an order $i \in \mathcal{I}_j$, the long-term accumulated reward of this matching is represented as the immediate reward of fulfilling the order $i$ and the future effect of repositioning the driver $j$ from $s = (l_j, t_j)$ to $s' = (l'_i, t_j + T_i)$:

$$v_\pi(i, j) = p_i a_i - p_i \beta + \gamma(V_\pi(t_j + T_i, l'_i) - V_\pi(t_j, l_j)), \tag{4}$$

where $\gamma$ is a discount factor which implies the weight of future effect, $T_i$ is the estimated travel time of order $i$.

In each time period, the objective of order dispatch is to find the optimal dispatch strategy $b$, so that the total value of all the dispatch is maximized. Recall that we denote a dispatch strategy of a driver $j$ as $b_j = \langle b_{ji} \rangle, i \in \mathcal{I}_j$. Let $b = \langle b_j \rangle, j \in \mathcal{J}$ denote the dispatch strategy of all the orders and drivers. Thus, we have the following integer linear program (ILP):

$$\max_b \quad \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} v_\pi(i, j) b_{ji} \tag{5}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}_i} b_{ji} \leq 1, \ \forall i \in \mathcal{I} \tag{6}$$

$$\sum_{i \in \mathcal{I}_j} b_{ji} \leq 1, \ \forall j \in \mathcal{J} \tag{7}$$

$$b_{ji} \in \{0, 1\}, \ \forall i \in \mathcal{I}, j \in \mathcal{J} \tag{8}$$

Constraint (6) indicates that at most one driver can be assigned to an order, where $\mathcal{J}_i$ is the set of feasible drivers that are within the dispatching distance of the order. Constraint (7) specifies that a driver can be assigned to at most one order. Constraint (8) restricts that all decision variables are binary. We can see that this problem belongs to the class of bipartite matching problems. In practice, we solve this problem with the Kuhn-Munkres (KM) algorithm [35].

### C. Revisiting Value Function of Request Pricing

When order dispatch is taken into account, the *immediate reward* (i.e., driver income in the current setting) in Eq. (1) can be rewritten as

$$r_\pi(x_i, a_i) = r(x_i, a_i; s_j, b_j) = \sum_{j \in \mathcal{J}_i} f(x_i, a_i) b_{ji}(p_i a_i - p_i \beta)$$

With a driver's spatio-temporal value definition in Eq. (3), the expected cumulative *future reward* of a certain pricing strategy $a_i$ towards request $i$ is:

$$R_\pi(x_i, a_i) = \gamma \sum_{j \in \mathcal{J}_i} f(x_i, a_i) b_{ji} (V_\pi(l_i', t_j + T_i) - V_\pi(l_j, t_j))$$

Combining the above two equations, the total expected reward of a pricing strategy $a_i$ is:

$$\begin{aligned} u_\pi(x_i, a_i) = \sum_{j \in \mathcal{J}_i} f(x_i, a_i) b_{ji} [p_i a_i - p_i \beta \\ + \gamma (V_\pi(l_i', t_j + T_i) - V_\pi(l_j, t_j))] \end{aligned} \tag{9}$$

From this equation, we can see that both the immediate and future rewards of a pricing strategy $a_i$ are also closely dependent on the dispatch strategy $b_j$ on the current time period, and the joint pricing and dispatch policy $\pi$ in future time periods. We refer to Table I as the list of notations.

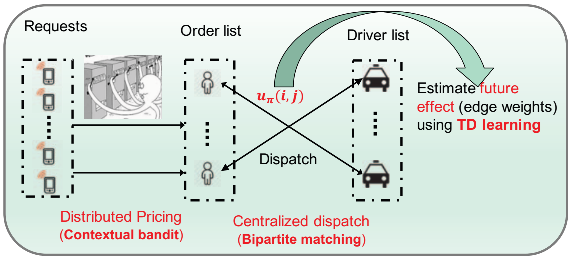
TABLE I
LIST OF NOTATIONS.

| | |
|---|---|
| $i$ | A request $i$ |
| $x_i$ | Context information of request $i$ |
| $p_i$ | Base price of request $i$ |
| $a_i$ | Pricing factor & an arm in the bandit algorithm |
| $f(x_i, a_i)$ | Request conversion rate |
| $j$ | A driver $j$ |
| $s_j$ | $s_j = (l_j, t_j)$: the spatio-temporal state of driver $j$ |
| $b_{ji}$ | Implies whether $j$ is assigned to $i$ |
| $\pi$ | $\pi = (\pi_p, \pi_d)$: the joint pricing and dispatch policy |
| $V_\pi(s_j)$ | Value function of driver $j$ given $s_j$ under policy $\pi$ |
| $v_\pi(i, j)$ | Value of a matching between $j$ and $i$ |
| $r_\pi(x_i, a_i)$ | Immediate reward of $a_i$ given $x_i$ under policy $\pi$ |
| $R_\pi(x_i, a_i)$ | Future effect of price $a_i$ given $x_i$ under policy $\pi$ |
| $u_\pi(x_i, a_i)$ | Total reward of price $a_i$ given $x_i$ under policy $\pi$ |

Despite the clear formulations for the value function of request pricing (Eq. (9)) and the global order dispatch optimization (Eqs. (5)-(8)), the two problems cannot be easily solved because of the unknown spatio-temporal value function $V_\pi(s)$ of a driver as well as the request conversion function $f(x, a)$. The inter-dependent pricing and dispatch policies make the learning of these values even more challenging. While reinforcement learning approaches have been proved to be effective in solving sequential decision making problems, they usually rely on a uniform MDP definition, which however, is not appropriate in our problem. This is because: (i) Request pricing needs to be done immediately after its emergence, making it impractical to formulate the state for the entire set of requests and drivers. (ii) If we define state for only individual requests, the decision entities would change from requests to drivers when performing a dispatch task. (iii) Moreover, even if we have a uniform MDP definition without considering the timing requirements of request pricing, the state and action spaces of the formulated MDP would be huge, making it impossible to be solved with current available techniques towards solving large scale MDPs. To address these challenges, we propose the InBEDE algorithm.

## V. Methodology

In this section, we present our proposed algorithm InBEDE to the above joint pricing and dispatch optimization problem. Figure 2 depicts the overall framework of InBEDE. The key idea of InBEDE is that, we use a contextual bandit algorithm for request pricing and employ a TD learning framework to estimate the future effect of pricing which is dependent on the global order dispatch component. In this way, the request pricing and order dispatch components are integrated, and the contextual bandit algorithm and the driver spatio-temporal value functions can be iteratively trained in a mutually bootstrapping manner.



Requests
Order list
Driver list
$u_\pi(i,j)$
Estimate future effect (edge weights) using TD learning
Dispatch
Distributed Pricing (Contextual bandit)
Centralized dispatch (Bipartite matching)

Fig. 2. An integrated approach for request pricing and order dispatch. Dispatch edge weights are updated every day. Bandit policy parameters are updated per dispatch step.

### A. Semi-Contextual Bandit for Distributed Request Pricing

Due to its ability to exploit an optimal action while exploring potentially more optimal actions (and updating policies *on the go*) based on the current context, we adopt the idea of a contextual bandit algorithm as the keystone for the distributed request pricing component. More specifically, we design a *semi-contextual bandit* algorithm, where our policy treats the emergence of a request $i$ as a *trial*.[2] In trial $i$, the *context* is the feature vector $x_i$ of the request which summarizes the request's contextual information described in Section IV-A.

Based on the context observed, a contextual bandit algorithm seeks to select an *arm* $a_i \in \mathcal{A}$ to maximize the expected long-term payoff. The *payoff* function $u_\pi(x_i, a_i)$ is defined as the reward associated with a certain arm $a_i$ and context $x_i$. Note that opposed to traditional contextual bandit algorithms where the decision is *one-shot* and the payoff function is merely the *immediate reward*, the payoff function in our semi-contextual bandit algorithm is a sum of the immediate reward and the future reward (as in Eq. (9)). Moreover, the payoff function in this problem is not only dependent on the pricing policy $\pi_p$, but also on the dispatch policy $\pi_d$. Formally, we define the expected payoff function of a certain joint pricing and dispatch policy $\pi = (\pi_p, \pi_d)$ as:

$$U_\pi(\mathcal{X}) = \mathbb{E}\Big\{ \sum_{x \in \mathcal{X}} u_\pi(x, a) \Big\}. \tag{10}$$

[2] Note that by treating each request as a trial, we assume that the pricing of different requests does not influence each other. While this assumption may not hold in some cases (e.g., for requests that are geographically close), it is natural that for the vast majority of the requests, the pricing of one request has very limited effect on one another.

Note that in the following of this sub-section, we omitted the sub-script $i$ of $x_i$ and $a_i$ for ease of representation. $\mathcal{X}$ is the total set of requests, and $u_\pi(x, a)$ is the payoff of selecting arm $a$ given context $x$ (see Eq. (9)).

As introduced above, there are several classical algorithms designed for contextual bandit problems [26], [27], [29], [30]. Without loss of generality, we adopt the LinUCB style algorithm due to its simplicity in implementation. Note that all the other algorithms can also be utilized in our framework. Similar to LinUCB, we assume for each trial, the expected payoff of an arm $a \in \mathcal{A}$ is a linear function in its $d$-dimensional context vector $x$ with parameter $\theta_a$:

$$\mathbb{E}\Big\{u_\pi(x,a)|x\Big\} = x^T\theta_a \tag{11}$$

To estimate $\theta_a$ for each arm $a$, we need to collect a set of context vectors $x$ with its corresponding payoff $u_\pi(x, a)$. Denote the training inputs before the current trial as a $m \times d$ matrix $D_a$, whose rows correspond to the $m$ training inputs (contexts) that observed before the current trial for the arm $a$, and let $c_a \in \mathrm{R}^m$ be the corresponding payoff vector. $\theta_a$ can be estimated using the ridge regression (as a closed-form solution) [36]:

$$\hat{\theta}_a = (D_a^T D_a + I_d)^{-1} D_a^T c_a, \tag{12}$$

where $I_d$ is a $d \times d$ identity matrix. Action is then selected as:

$$a = \arg\max_{a \in \mathcal{A}} (x^T \hat{\theta}_a + \alpha\sqrt{x^T A_a^{-1} x}), \tag{13}$$

where $A_a = D_a^T D_a + I_d$, $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$ is a constant with $\delta > 0$.

### B. Estimating Future Effect of Pricing with TD Learning

As is noticed, in the above semi-contextual bandit algorithm, the future effect $R_\pi(x_i, a_i)$ of a currently selected arm $a_i$ (i.e., a pricing strategy) cannot be known immediately, since we need to know the future spatio-temporal value $V_\pi(s_j) = V_\pi(l_j, t_j)$ of the driver $j$ that is assigned. This prevents us from learning and updating our contextual bandit policy online. To overcome this problem, we integrate our semi-contextual bandit algorithm with TD learning, where instead of getting the exact long-term action value using the Monte Carlo method, we obtain an approximation of the value by way of dynamic programming (DP).

More specifically, our approximation of the current pricing strategy $a_i$ is a sum of the immediate reward and an estimated future effect of repositioning the assigned driver:

$$\begin{aligned}\hat{u}_\pi(x_i, a_i) = \sum_{i \in \mathcal{J}_i} & f(x_i, a_i) b_{ji} [p_i a_i - p_i \beta \\ & + \gamma(\hat{V}_\pi(l'_i, t_j + T_i; \phi) - \hat{V}_\pi(l_j, t_j; \phi))]\end{aligned} \tag{14}$$

where $\hat{V}_\pi(l_j, t_j; \phi)$ is an approximation of the long term spatio-temporal value of a driver with parameter $\phi$. Such approximation can be achieved by previous approaches such as the tabular [8] and deep neural network [9] approximators. We adopt the latter due to its superior power of value representation. Note that the driver spatio-temporal value approximator is a sub-routine of InBEDE, and InBEDE can also be easily adapted to the tabular approximator. With Eq. (14), once we know the request conversion result (which replaces $f(x_i, a_i)$ by either 1 or 0) and the dispatch strategies $b_{ji}, \forall j \in \mathcal{J}_i$ that are related to request $i$, we are then able to approximate the long term value of a current pricing strategy $a_i$.

### C. InBEDE

As shown in Algorithm 1, InBEDE proceeds in an iterative manner, where the input is the initial order list $OL_0$ and driver list $DL_0$ at time $t = 0$. It starts with an initialization of the bandit algorithm parameters $\theta$ and the driver value network approximator parameters $\phi$. It then enters the iterative training loop in Lines 3-16. Within each iteration loop (usually a day), it goes through all the order dispatch time slots $t = 0, \ldots, T$. For each $t$, it first obtains the updated order list $OL_t$ and driver list $DL_t$, it then employs the current contextual bandit algorithm with parameter $\theta$ to price the requests that arrive within the time slot $t$ (Lines 6-11). At the end of time slot $t$, the bandit parameters $\theta$ are updated with the immediate reward $r(x_i, a_i)$ and TD estimation of future reward. After an iteration of dispatch is finished (end of a day), the driver trajectories are collected and the parameters $\phi$ of the driver spatio-temporal value approximators are updated with a subroutine [9].

**Algorithm 1:** InBEDE

```
1  Input: order list $OL_0$ at $t = 0$, idle driver list $DL_0$ at
     $t = 0$;
2  Initialize $\theta, \phi$;
3  for Iteration $1, \ldots,$ do
4      for $t = 0, \ldots, T$ do
5          Update $OL_t$ and $DL_t$;
6          for Request $i$ that arrives at $t$ do
7              Observe feature $x_i$;
8              Select an arm $a_i$ according to the bandit
                 algorithm $\theta$;
9              if Request $i$ converts into order then
10                 Append request $i$ into order list $OL_t$;
11         Perform dispatch among $OL_t$ and $DL_t$ with
             Eqs. (5)-(8), using the driver value network $\phi$;
12         for Request $i$ that arrives within $t$ do
13             Get estimated reward $\hat{u}_\pi(x_i, a_i)$ using Eq. (14);
14             Update parameter $\theta$ with $(x_i, a_i, \hat{u}_\pi(x_i, a_i))$;
15     Collect driver trajectories $(s, a', s', r)$;
16     Update parameter $\phi$ with the collected driver
         trajectories;
17 return $\theta, \phi$;
```

## VI. Experimental Evaluation

In this section, we evaluate our proposed InBEDE algorithm using a system simulator of Didi Chuxing with real-world data from two major cities in China. City A is a median-sized city in China with a population of around $5 - 10$ millions, while city B is a large city with a population of over $10$ millions.[3]

[3] The city names are not revealed per company data policy.

### A. *Environment Setting and Simulator Description*

We first describe the dataset and the dedicated ride-hailing system simulator utilized in the evaluation.

*1) Dataset Description:* To evaluate the performances of our proposed approach, we randomly select a date for each of the two cities for evaluation (Oct. 25, 2017 for city A, and Mar. 5, 2018 for city B).[4] The anonymous dataset used in this paper consists of the following components:

*Request data.* Each entry of the request data contains the contextual information of the request (as introduced in Section IV-A) and whether it converts to an order. For the two selected dates, there are in total around $200,000$ requests for city A and $1,000,000$ requests for city B.

*Driver log on and off activities data.* This dataset records the time and location of a driver when he/she logs on/off the ride-hailing platform. This dataset is utilized to simulate the working status of the drivers.

*Driver trajectory data.* This includes both cases where drivers are serving a passenger (on service) and driving idly without fulfilling a trip order. This dataset is used to simulate the driving behavior of the drivers when they are logged in the ride-hailing platform.

*2) System Simulator:* For all the experiments, the feasible action space is $\mathcal{A} = \{0.85,\ 0.9,\ 0.95,\ 1,\ 1.05,\ 1.1,\ 1.15\}$. The unit time period of order dispatch is set as 2 seconds, which is consistent with typical real-world practice.

We use a system simulator based on real-world data from Didi Chuxing for the evaluation. The differences between the simulated results and the real-world situation is less than $2\%$, in terms of core metrics such as drivers' income, order answer rate and driver idle rate, etc. The simulator consists of the following three modules:

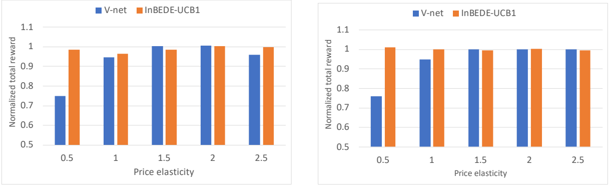
*a) Request conversion module.:* This module takes as input a request and outputs whether it converts into an order. We simulate it with two types of CR functions. (i) *Context-Free CR function.* In the first type, we evaluate our approach under a context-free CR function of the form $f(x, a) = f_0 + k(a-1)$, where $f_0$ is a base probability when the pricing factor $a = 1$, $k$ is a price elasticity factor implying a passenger's sensitivity to prices. In this case, the CR function only depends on the pricing strategy $a$. (ii) *Contextual CR Function.* The second type of CR function is designed based on a logistic regression (LR) model learned from historical request data. This model is widely adopted in existing taxi pricing and dispatch literature [37], [38]. To get the request conversion result for a given pricing factor under this LR model, a naive way is just to multiply the base price by the pricing factor, and feed the new price to the learned LR model. However, it is worth noting that, all the price features in the learned LR model are only original prices, while in practice, the weights of the price are much larger than the learned values, as customers are always more sensitive to the "changes" of prices than to the absolute values of the prices. In order to characterize this phenomenon, we modify the learned LR model by multiplying the weight of the price feature by a factor.

*b) Order dispatch module.:* The core of this module is the KM algorithm, which solves the ILP in Eqs. (5)-(8). It takes as input values $v_\pi(i, j)$ of matching from driver $j$ to order $i$, and outputs the optimal matching strategy $b$.

*c) Driver status update module.:* This module updates the location and vacancy status (vacant, busy, or on the way to pick up a passenger) of a driver after order dispatch.

### B. *Context-Free CR Function*

To see whether the bandit algorithm works in the request pricing tasks, we first evaluate the semi-bandit algorithm in a simplified environment setting, where 1) the CR function is context-free of the form $f(x_i, a_i) = f_0 + k(a_i - 1)$ (as described in the previous subsection), 2) we use a pre-trained (using the original prices in the historical data as the pricing strategy, trained with one month of driver trajectory data) and fixed driver spatio-temporal value network $V_\pi(t, l)$. We fix $f_0$ as $0.5$ and $0.3$ respectively for cities A and B which are close to the average CR according to our request dataset. We then vary $k$ to evaluate the performance of different approaches under different price elasticity values.



(a) City A

(b) City B

Fig. 3. Total reward of different methods on a simplified context-free CR function setting, normalized by dividing the reward of the Optimal approach.

We compare the following three methods:

- V-net, which is the state-of-the-art order dispatch approach [9]. This approach only optimizes dispatch strategy, and uses original prices in the historical data as the pricing strategy.
- InBEDE-UCB1. UCB1 is a context-free bandit algorithm. Since the CR function is context-free, we use UCB1 as the bandit algorithm component.
- Optimal-Fixed-Price. For context-free CR function, request conversion rate only depends on request prices. Hence, we can find the optimal fixed pricing strategy by a brute-force search in the price space $\mathcal{A}$. This approach also uses V-net as the order dispatch component. Note that under the context-free CR function, this strategy is close-to-optimal.

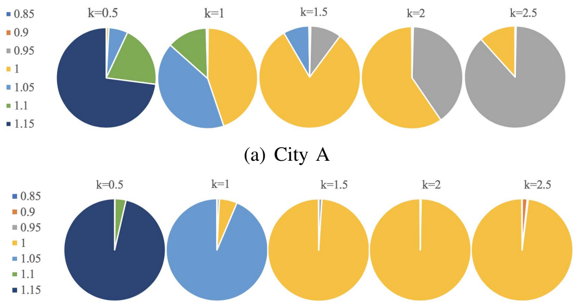
Figure 3 shows the comparison results, where the $x$-axis is the price elasticity $k$, and the $y$-axis is the normalized total reward of V-net and InBEDE-UCB1 with respect to the Optimal-Fixed-Price approach, so that by default, the total reward of the Optimal-Fixed-Price approach is 1. We can

[4] Despite only one day data, there are already more than 200,000 requests for city A and 1,000,000 requests for city B. The sizes of the datasets are large enough to demonstrate the performances of our approaches.

see that for both cities and different price elasticity $k$, our algorithm significantly outperforms the V-net baseline, and is close to the Optimal-Fixed-Price approach. It is worth noting that in some cases (e.g., for City A when price elasticity $k = 1.5$), our InBEDE algorithm obtains smaller total reward than the V-net baseline. This is because under such a setting, the original prices (i.e., with a pricing factor of 1) used in the V-net baseline happen to be the optimal price, while InBEDE, which is a reinforcement learning method in essence, spends some trials to explore the optimal pricing strategies at the beginning. Thus, InBEDE cannot achieve optimality from the very beginning. However, the close-to-optimal performance of InBEDE shows that InBEDE is able to learn optimal pricing strategies after enough trials, and is able to generalize to different price elasticity settings. Similarly in Figure 3(b), the performances of the three methods are almost the same when $k = 1.5$ to $2.5$. This is because in this case, the optimal price is always the original price in City B. This will be further demonstrated in the following.

We now show in Figure 4 the price statistics selected by our InBEDE-UCB1 algorithm, where the area of a sector indicates the portion of the corresponding price. Comparing Figure 4(a) with Figure 4(b), we find that, the portion of high price factors in City B tends to be larger than that of City A. This is because the average request conversion probability (used as the base probability $f_0$ in the simulator) of City B is lower than that of City A, i.e., customers in City B have a same probability of sending out an order even with a slightly higher price.

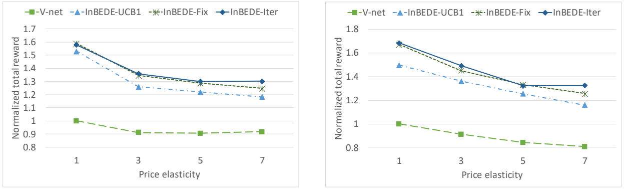
Another interesting finding is that when price elasticity $k$ increases from 0.5 to 2.5 (i.e., when passengers are more sensitive to price changes), the portion of lower prices selected by InBEDE-UCB1 also increases. This follows the intuition that when price elasticity is high (i.e., when customers are more sensitive to price changes), a lower price should be quoted. We will further elaborate on this finding in the following subsection, with experiments done on a contextual CR function learned on real-world historical request conversion data of the two underlying cities.



(a) City A

(b) City B

Fig. 4. Statistics of prices selected by InBEDE-UCB1 in cities A and B, with varying price elasticity values from 0.5 to 2.5.



(a) City A

(b) City B

Fig. 5. Total reward of different methods on a contextual CR function setting. The $x$-axis denotes the price elasticity $k$, and the $y$-axis denotes the total reward, which is normalized by dividing the total reward of the V-net method with $k = 1$.

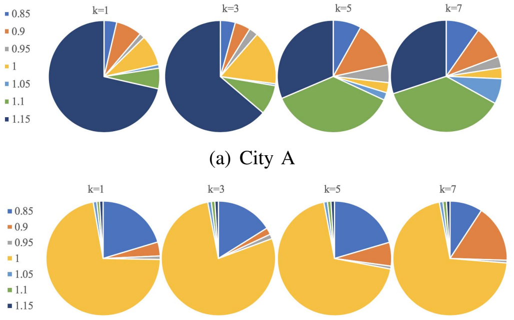
### *C. Contextual CR Function*

In this set of experiments, we evaluate our proposed algorithms under a more realistic scenario, where the contextual CR function (learned from historical request conversion data) is used as the request conversion module. We compare the following different algorithms:

- V-net.
- InBEDE-UCB1.
- InBEDE-Fix. This method uses a fixed driver spatio-temporal value network (pre-trained along with original pricing strategies) for order dispatch.
- InBEDE-Iter. This method uses the iteratively trained pricing (output by the bandit algorithm component) and dispatch policies (using the learned driver spatio-temporal values) in InBEDE, as shown in Algorithm 1.

The comparison results are shown in Figure 5. We can see that, first, the total reward obtained by all the approaches generally decreases when price elasticity increases. Intuitively, this is because that higher price elasticity implies that the demand from customers is somewhat less “rigid demand”, thus increased prices would lead to a higher loss of demand. Second, our proposed methods (i.e., InBEDE-UCB1, InBEDE-Fix and InBEDE-Iter) significantly outperform the V-net baseline. Third, InBEDE-Fix and InBEDE-Iter, which utilize LinUCB as the bandit algorithm, achieve much higher total reward than InBEDE-UCB1. This demonstrates that it is beneficial to exploit the contextual information while making pricing decisions. Last, with the iterative training of the contextual bandit model and the driver spatio-temporal value network, InBEDE-Iter is able to gain a considerably higher total reward (in most cases) than the pre-trained and fixed driver value network (InBEDE-Fix). This demonstrates that the performance of the joint pricing and dispatch could be further strengthened with the mutually bootstrapped iterative training. Note that despite the smaller amount compared with the improvement with respect to InBEDE-UCB1, this improvement is still considerable because even a small relative improvement would incur a great absolute gain, considering the millions of base total driver income per day.

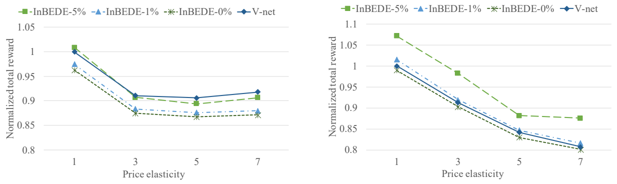
We also show price statistics of InBEDE-Iter in Figure 6. Similar to the contextual-free scenario, we can see that when customer price elasticity $k$ increases (from $k = 1$ to $k = 7$),



(a) City A

(b) City B

Fig. 6. Statistics of the price selected by InBEDE-Iter in the contextual CR function scenario, with varying price elasticity values from $k = 1$ to 7.



(a) City A

(b) City B

Fig. 7. Total reward obtained by different methods on a contextual CR function setting with price constraints, where the $x$-axis denotes the price elasticity $k$, and the $y$-axis denotes the total reward, which is normalized by dividing the reward of the V-net method with $k = 1$.

the portion of higher prices selected by our proposed approach significantly decreases. For example, in City A, the portion of pricing factor 1.15 decreases from $73.49\%$ to $56.71\%$, and the portion of pricing factor 0.85 increases from $5.66\%$ to $9.42\%$. More interestingly, since the CR function is fitted with real-world request conversion data, it demonstrates, somewhat counter-intuitively, that a potentially higher price should be quoted to requests in order to maximize the total reward, while in practice, the prices of most ride-hailing platforms are much lower. We conjecture that this is mainly due to the considerations of customer satisfaction and market competition.

### D. Constraints on Price Increase

From statistics of prices selected by our proposed InBEDE framework, we can see that in most cases, increasing prices would lead to increased total reward. However, increased prices would also incur a substantially lower customer satisfaction. As a result, we come up with a modified version of InBEDE, where the portions of increased prices are strictly constrained. More specifically, for each price factor which is larger that 1, we restrict that the portion of it being selected cannot exceed a certain threshold (e.g., $1\%, 5\%$).

**Running estimate trick.** With price constraints, it means that we cannot arbitrarily select any price factor merely with the bandit algorithm. Instead, we need to filter out the restricted portion of most "effective" requests where increased prices could return the highest "advantage" compared with the original price. However, since we do not know future request information, we cannot obtain the statistics of this advantage for the entire request population. To handle this issue, we use the lowest advantage (one for each increased price) of the previously selected increased prices as a *running estimate* of the lower bound advantage of the portion of effective requests. For an arm selected by the bandit algorithm, we first compare the advantage of this arm with the estimated lower bound, and finally select this arm only when its advantage is larger than the lower bound. Otherwise, we use the original price.

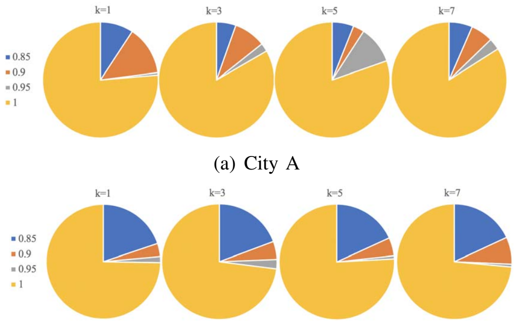
We evaluate three different extents of thresholds, i.e., $5\%, 1\%$, and $0\%$. E.g., $1\%$ means that the portion of each increased price (i.e., $1.05, 1.1$ and $1.15$) cannot exceed $1\%$ of the entire request population. Correspondingly, the total portion of increase prices cannot exceed $3\%$. $0\%$ means that strictly no price increase is allowed. We compare the above three conditions with the V-net approach.

The results are shown in Figure 7. For both cities, we see that following our expectation, with a more strict constraint on price increase (from $5\%$ to $0\%$), the total reward decreases correspondingly. For city A, the total reward obtained by InBEDE-$5\%$ is still close to that of V-net. When there is strictly no increase of prices, the total reward is on average $4.3\%$ lower than that of V-net. However, considering that there are an overall $19\%$ of requests with decreased prices (as shown in Figure 8(a)), this decrease is an acceptable amount.

As shown in Figure 7(b), the performance of InBEDE is even better on city B. It is worth mentioning that, when only a maximal $1\%$ price increase is allowed (and the prices of an overall $25.8\%$ of the requests are decreased), InBEDE is still able to obtain a substantial gain compared with V-net. Moreover, even when strictly no price increases are allowed, the total reward obtained by InBEDE is still comparable to that of the V-net method. Figure 8(b) shows the price statistics of InBEDE-$0\%$ on city B under various price elasticity values. Take $k = 1$ as an example, we can see that prices of the the majority of requests are unchanged. There is a portion of $19.86\%$ of requests whose prices are decreased to 0.85 of the original prices, and $3.69\%$ and $1.69\%$ of requests are imposed price decreases to 0.9 and 0.95, respectively, totalling $25.24\%$ of requests with price discounts. Combining a loss of only $1\%$ versus the V-net method in Figure 7(b), we can see that InBEDE with price constraints has been truly effective in maintaining total driver income while benefiting customers with a substantially decreased price.

## VII. Conclusion

Pricing and dispatch have been the two major leverages towards mitigating supply-demand imbalance of ride-hailing system. Recently, we have observed that pricing and dispatch decisions are not independent, but are inherently interrelated. Based on this observation, we make the first attempt to jointly optimize pricing and dispatch within a sophisticatedly designed, unified framework called InBEDE. In InBEDE, the two components, namely contextual bandit for request pricing and TD learning for order dispatch, are mutually bootstrapped via



(a) City A

(b) City B

Fig. 8. Statistics of the price selected by InBEDE with strictly no price increase. Price elasticity values are varied from $k = 1$ to 7. The prices of around $17\% \sim 27\%$ of the requests are decreased, which could generate a substantial benefit to the customers. We do not show price statistics for InBEDE-1% and InBEDE-5% due to space limit, but the patterns are similar.

an iterative training process. We demonstrate the effectiveness of InBEDE by testing it using real-world data from two Chinese cities on the Didi Chuxing platform.

## References

[1] Siddhartha Banerjee, Daniel Freund, and Thodoris Lykouris. Pricing and optimization in shared vehicle systems: An approximation framework. *arXiv preprint arXiv:1608.06819*, 2016.

[2] Mengjing Chen, Weiran Shen, Pingzhong Tang, and Song Zuo. Optimal vehicle dispatching schemes via dynamic pricing. *arXiv preprint arXiv:1707.01625*, 2017.

[3] Juan Camilo Castillo, Dan Knoepfle, and Glen Weyl. Surge pricing solves the wild goose chase. In *EC*, pages 241–242, 2017.

[4] Hongyao Ma, Fei Fang, and David C Parkes. Spatio-temporal pricing for ridesharing platforms. *arXiv preprint arXiv:1801.04015*, 2018.

[5] Ziqi Liao. Real-time taxi dispatching using global positioning systems. *Communications of the ACM*, 46(5):81–83, 2003.

[6] Rick Zhang and Marco Pavone. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. *The International Journal of Robotics Research*, 35(1-3):186–203, 2016.

[7] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. A taxi order dispatch model based on combinatorial optimization. In *SIGKDD*, pages 2151–2159, 2017.

[8] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *SIGKDD*, pages 905–913, 2018.

[9] Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A deep value-network based approach for multi-driver order dispatching. In *SIGKDD*, 2019.

[10] Nikita Korolko, Dawn Woodard, Chiwei Yan, and Helin Zhu. Dynamic pricing and matching in ride-hailing platforms. *Available at SSRN*, 2018.

[11] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[13] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[16] Hai Yang, CS Fung, Ka Io Wong, and Sze Chun Wong. Nonlinear pricing of taxi services. *Transportation Research Part A: Policy and Practice*, 44(5):337–348, 2010.

[17] Siddhartha Banerjee, Carlos Riquelme, and Ramesh Johari. Pricing in ride-share platforms: A queueing-theoretic approach. *SSRN 2568258*, 2015.

[18] Jiaru Bai, Kut C So, Christopher S Tang, Xiqun Chen, and Hai Wang. Coordinating supply and demand on an on-demand service platform with impatient customers. *Manufacturing & Service Operations Management*, 2018.

[19] Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *ICDM*, 2018.

[20] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *WWW*, pages 983–994, 2019.

[21] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[22] Apostolos N Burnetas and Michael N Katehakis. Optimal adaptive policies for sequential allocation problems. *Advances in Applied Mathematics*, 17(2):122–142, 1996.

[23] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[24] Steven L Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, pages 661–670, 2010.

[27] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *ICML*, pages 127–135, 2013.

[28] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*, pages 1015–1022, 2010.

[29] Michal Valko, Nathaniel Korda, Rémi Munos, Ilias Flaounas, and Nelo Cristianini. Finite-time analysis of kernelised contextual bandits. *arXiv preprint arXiv:1309.6869*, 2013.

[30] Robin Allesiardo, Raphaël Féraud, and Djallel Bouneffouf. A neural networks committee for the contextual bandit problem. In *NIPS*, pages 374–381, 2014.

[31] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. Collaborative filtering bandits. In *SIGIR*, pages 539–548, 2016.

[32] Nathan Korda, Balázs Szörényi, and Li Shuai. Distributed clustering of linear bandits in peer to peer networks. In *Journal of machine learning research workshop and conference proceedings*, volume 48, pages 1301–1309. International Machine Learning Societ, 2016.

[33] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[35] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[36] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[37] M Keith Chen and Michael Sheldon. Dynamic pricing in a labor market: Surge pricing and flexible work on the uber platform. In *EC*, page 455, 2016.

[38] Peter Cohen, Robert Hahn, Jonathan Hall, Steven Levitt, and Robert Metcalfe. Using big data to estimate consumer surplus: The case of uber. Technical report, National Bureau of Economic Research, 2016.