8-2021

Learning to assign: Towards fair task assignment in large-scale ride hailing

Dingyuan SHI

Yongxin TONG

Zimu ZHOU
Singapore Management University, zimuzhou@smu.edu.sg

Bingchen SONG

Weifeng LV

See next page for additional authors

Learning to Assign: Towards Fair Task Assignment in Large-Scale Ride Hailing

Dingyuan Shi
SKLSDE & BDBC, Beihang University
chnsdy@buaa.edu.cn

Yongxin Tong
SKLSDE & BDBC, Beihang University
yxtong@buaa.edu.cn

Zimu Zhou
Singapore Management University
zimuzhou@smu.edu.sg

Bingchen Song
SKLSDE & BDBC, Beihang University
songbch@buaa.edu.cn

Weifeng Lv
SKLSDE & BDBC, Beihang University
lwf@buaa.edu.cn

Qiang Yang
The Hong Kong University of Science and Technology
AI Group, WeBank Co., Ltd
qyang@cse.ust.hk

ABSTRACT

Ride hailing is a widespread shared mobility application where the central issue is to assign taxi requests to drivers with various objectives. Despite extensive research on task assignment in ride hailing, the fairness of earnings among drivers is largely neglected. Pioneer studies on fair task assignment in ride hailing are ineffective and inefficient due to their myopic optimization perspective and time-consuming assignment techniques. In this work, we propose LAF, an effective and efficient task assignment scheme that optimizes both utility and fairness. We adopt reinforcement learning to make assignments in a holistic manner and propose a set of acceleration techniques to enable fast fair assignment on large-scale data. Experiments show that LAF outperforms the state-of-the-arts by up to 86.7%, 29.1%, 797% on fairness, utility and efficiency, respectively.

CCS CONCEPTS

• **Applied computing** → *Economics.*

KEYWORDS

Fairness; Ride Hailing; Reinforcement Learning

ACM Reference Format:
Dingyuan Shi, Yongxin Tong, Zimu Zhou, Bingchen Song, Weifeng Lv, and Qiang Yang. 2021. Learning to Assign: Towards Fair Task Assignment in Large-Scale Ride Hailing. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Ride hailing is an emerging shared mobility service that significantly increases the urban traffic capacity. For example, a major ride hailing platform in China has reached over 50 million daily trips in 2020 [1]. A ride hailing service consists of drivers, passengers, and the platform, where the platform matches taxi requests submitted by passengers to drivers. Matching between taxi requests and drivers, a.k.a. *task assignment*, is the core issue in ride hailing research [19, 21, 23, 24, 26].

Existing studies mainly optimize task assignment from the perspective of the *platform* or the *passengers*, whereas few have investigated the problem from the *drivers'* perspective. For example, many assignment algorithms have been designed to maximize the utility [19, 23, 26] or minimize the travel cost [21] or the passengers' waiting time [24]. Only very recently did pioneer work [13, 17] explore objectives such as balancing earnings among drivers during task assignment. We argue that blindly assigning drivers to tasks that maximizes utility may impair the drivers' experience in terms of notable earning gaps among drivers[2]. Fig. 1 simulates the trajectories of four drivers via an assignment algorithm that merely maximizes utility. Driver 1 is first assigned a request which ends in the remote area. After completing this order, it is difficult for him/her to get back to the busy area, leading to low hourly earnings. In contrast, driver 3 is assigned by chance to another request which ends in the busy area so that he/she can have higher hourly earnings. Driver 2 stays in the busy area but is rarely assigned a request. The long idle time also leads to low hourly earnings. In comparison, driver 4 happened to be assigned many requests and the short idle time contributes to a higher hourly earning. Such gaps in hourly earnings among drivers (driver 1, 2 vs. driver 3, 4 in this example) may discourage drivers and result in *unfair* task assignment.

Although fairness in task assignment has been extensively researched in application domains such as load balancing in cloud computing [4, 6, 10, 12] and kidney exchange [7, 15], fair task assignment in ride hailing faces unique challenges. *(i)* online setting: ride hailing is a two-sided online assignment scenario, with high spatiotemporal dependencies and variations [20]. *(ii)* bi-objective optimization: an ideal assignment algorithm for ride hailing should optimize both the utility and the fairness among drivers, under various practical constraints. *(iii)* high efficiency requirement: real-world ride hailing platforms require fast assignments on data at urban-scale. The state-of-the-art fair assignment algorithms for ride hailing [13, 17] fail to address all these challenges. Firstly, these

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/1122445.1122456

[1] https://www.didiglobal.com/news/newsDetail?id=955&type=news

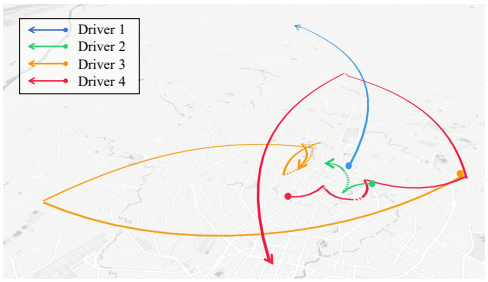


Figure 1: Simulated driver trajectories. A solid (dash) line means the driver is serving a request (idle).

solutions are myopic. That is, they ignore the impact of the current assignment to future ones, which largely shrinks the optimization space and thus notably reduces the achievable utility and fairness. Secondly, they either rely on linear programming [17] or require multiple rounds of re-assignments [13], making them inefficient for real-time responses over large-scale data.

In this paper, we propose Learning to Assign with Fairness (LAF), an effective and efficient task assignment scheme that optimizes both utility (measured by expected total earnings of all drivers) and fairness (measured by temporal earnings fairness among drivers) in ride hailing. LAF explicitly accounts for the dependencies among assignments by learning *future-aware* assigning strategies via reinforcement learning. Such learned assigning strategies can optimize utility and fairness in a holistic manner. For efficient assignment, LAF embeds fairness and utility optimization into the same augmentation operation, and takes advantages of the sparsity in the bipartite graph for further acceleration. It is also worth mentioning that we propose a *weighted* amortized fairness metric rather than the conventional *unweighted* one [17] to characterize the drivers' earnings fairness at a finer time granularity *e.g.*, hourly earnings fairness.

Evaluations show an improvement of 45.7%~86.7% and 7.7%~29.1% on fairness and utility, respectively, over the state-of-the-arts [13, 17]. Our LAF also runs up to 700× faster than [13, 17]. The main contributions of this work are summarized as follows.

- To the best of our knowledge, this is the first work that explicitly considers the dependency between current and future assignments to improve the performance of fair task assignment in ride hailing.
- We propose LAF, a novel reinforcement learning based fair task assignment scheme that is adaptive to highly dynamic traffic, aligned with practical settings, and fit for large-scale ride hailing applications.
- Extensive evaluations show our LAF outperforms the state-of-the-arts [13, 17] by a large margin in terms of fairness, utility, and efficiency.

In the rest of this paper, we review related work in Sec. 2, introduce our problem in Sec. 3 and explain our methods in Sec. 4. We present the evaluations in Sec. 5 and conclude in Sec. 6.

2 RELATED WORK

Our work is mainly related to two threads of research: *task assignment in ride hailing* and *fairness in assignment problems.*

Task Assignment in Ride Hailing. Due to its fast expansion in metropolis, ride hailing has attracted extensive research interests [1, 21, 23, 24, 26], where a core topic is to assign taxi requests to drivers for certain optimization objects. Common goals in ride hailing include maximizing profit [23, 26], minimizing travel cost [21, 22], minimizing passengers' waiting time [1, 24], and so on.

Many task assignment algorithms focus on providing theoretical performance guarantees [24]. However, they often make assumptions such as independence among drivers and assignments, which often prohibits them from delivering the expected performance in real-world applications. A promising alternative is to automatically learn to optimize the assignments from historical data with few assumptions. For example, some recent work [25] achieves the state-of-the-art performance by using reinforcement learning to optimize utility. In this work, we also target at practical task assignment algorithms suited for large-scale applications. Our method is inspired by [25] but we are the first to design a reinforcement learning based solution that optimizes both utility and fairness.

Fairness in Assignment Problems. Fairness has long been an essential factor of assignment problems in various applications [9, 14]. Research on fair assignment can be divided into two categories depending on whether the tasks and workers are static or dynamic.

In static fair task assignment, both workers and tasks are static, which is common in crowdsourcing [3, 11] and kidney exchange [7, 15]. Dynamic fair assignment is more difficult, where one or both side can be dynamic. This setting is fit for applications such as cloud computing [12] and web request allocation [4], where the objective is to balance the load of servers. Some work [5] studied fair assignment in spatial crowdsourcing, where fairness means workers serve the same number or value of tasks. This fairness goal is unfit for ride hailing, since different taxi drivers have different working times.

Fairness in ride hailing belongs to dynamic fair assignment problems. Of particular interest is the fairness of driver earnings [13, 17]. Fair assignment in ride hailing is challenging due to the strong spatiotemporal dynamics of tasks and workers. In this work, we propose a new driver earnings fairness metrics that accounts for such spatiotemporal dynamics and devise efficient and effective assignment algorithms that notably outperform the state-of-the-art fair assignment algorithms [13, 17].

3 PROBLEM STATEMENT

As with previous studies [13, 17], we consider *batched* assignments of drivers to requests. Let W and R be the sets of drivers and requests in the time horizon of interest T. In each batch t, assignments between the requests $R^{(t)}$ and drivers $W^{(t)}$ available in this batch are formulated as a bipartite matching problem to optimize both *utility* and *fairness*. The batched setting is widely adopted in real-world large-scale ride hailing applications [19, 25, 26].

Definition 1 (Request). *A taxi request $r \in R$ is denoted as a tuple $< o_r, d_r, p_r, \tau_r >$, where o_r, d_r, p_r and τ_r represent the origin, destination, price and duration of r.*

In real-world ride hailing applications, the origin and the destination of a request are input by the passengers and uploaded to the platform. We make no assumptions on the origin and the destination of a request. Upon receiving the request, the platform determines the price and estimates the duration for the request based on the trip distance and other factors *e.g.*, traffic congestion. For simplicity, we assume the duration is a multiple of t. Note that the ratio p_r/τ_r decides the per-batch earnings of a driver if the request is assigned to him/her, as we will explain next.

Definition 2 (Driver). *A taxi driver $w \in W$ is denoted as a tuple $< l_w^{(t)}, u_w^{(t)}, a_w^{(t)} >$, where $l_w^{(t)}$, $u_w^{(t)}$, and $a_w^{(t)}$ are w's current location, earnings and status in batch t.*

The status $a_w^{(t)}$ is a binary indicator of 0 or 1. $a_w^{(t)} = 0$ if the driver is inactive, *i.e.*, not on the platform. $a_w^{(t)} = 1$ if the driver is active, which can be either idle or serving a request. We assume $a_w^{(t)}$ stays the same within a batch since the batch size is small in practice (*e.g.*, 2 seconds [26]). The earnings $u_w^{(t)} = 0$ if the driver is idle, whereas $u_w^{(t)} = p_r/\tau_r$ if he/she is serving request r in batch t. Unlike prior studies [21, 24] that assume *independent* driver distributions across batches, we consider a more realistic *dependency* driver setting. That is, the driver distribution in future batches is influenced by that of the current batch as well as the current assignment. The *dependency* driver setting accounts for the *temporal dependency of assignments across batches.*

Definition 3 (Bipartite Graph). *We use a bipartite graph $G^{(t)} =< R^{(t)}, W^{(t)}, E^{(t)} >$ to denote the candidate assignments between drivers and requests in bath t, where node sets $R^{(t)}$ and $W^{(t)}$ are the requests to be assigned and the available drivers in t, respectively. There is an edge $(r, w) \in E^{(t)}$ with a weight $\theta_{r,w}$ if request r can be assigned to driver w.*

In line with prior research on task assignment in ride hailing [26], an edge only exists if the request-driver distance is within a threshold to avoid long waiting time of passengers. We also assign a rejection rate $\lambda_{r,w}$ to each edge (r, w) to account for other user experience related factors. The distance threshold and the rejection rates are set by the platform. The weight $\theta_{r,w}$ is initially set as the price p_r of request r.

Our objective is to find for each batch $t \in \{1, 2, \cdots, T\}$ a matching $M^{(t)}$ on candidate assignments $G^{(t)}$ that optimize *total utility* and *temporal earnings fairness* defined as below.

Definition 4 (Total Utility). *Given the reusable set of drivers W and the dynamically appeared set of requests R, the total utility is the expected accumulated earnings across all drivers till batch T,* i.e.,

$$U = \sum_{w \in W} \mathbb{E}\left[\sum_{t \in T} u_w^{(t)}\right] \tag{1}$$

where $\mathbb{E}[.]$ takes the expectation. The expected accumulative earnings $\mathbb{E}\left[\sum_{t \in T} u_w^{(t)}\right]$ of w are decided by the matching results:

$$\mathbb{E}\left[\sum_{t \in T} u_w^{(t)}\right] = \sum_{t \in T} \sum_{(r,w) \in M^{(t)}} (1 - \lambda_{w,r}) \cdot p_r \tag{2}$$

We maximize U to optimize the overall earnings of drivers.

Table 1: Summary of important notations.

Notation	Description
T	time horizon of interest *i.e.*, # batches, at the scale of a day
t	a batch, duration of a batch at the scale of seconds
W, R	set of all drivers and requests
$W^{(t)}, R^{(t)}$	set of available drivers and requests in batch t
$l_w^{(t)}, u_w^{(t)}, a_w^{(t)}$	location, earnings and status of w in batch t
o_r, d_r, p_r, τ_r	origin, destination, price, and duration of request r
$G^{(t)}$	bipartite graph of candidate assignments in batch t
$E^{(t)}$	candidate assignments in batch t
$M^{(t)}$	actual assignments in batch t
$\lambda_{w,r}$	rejection rate of edge (w, r), *i.e.*, assigning r to w
$\theta_{w,r}$	weight of edge (w, r)
U	total utility
F_w	weighted amortized fairness
$\xi^{(t)}$	weight for active time in F_w
F	temporal earnings fairness
V	value function, $V(l)$ denotes the value in location l
H, S	hexagon and square value functions
$\mathcal{S}$	guidance scheme for idle drivers

We adopt the notion of *amortized fairness* [17] to characterize the *temporal earnings fairness* among drivers. The amortized fairness, which is the accumulative earnings over the active time $(\sum_{t \in T} u_w^{(t)})/(\sum_{t \in T} a_w^{(t)})$, should be equal among drivers. However, we argue that the naive amortized fairness can be unfair to drivers who have different working hour preferences or schedules (*e.g.*, daytime, night hours, or rush hours). Since the taxi demand in different time of the day varies, the potential earnings also differs. Therefore, enforcing equal earnings proportional to the active time leads to earnings inequality between drivers *e.g.*, who mainly work in the day and those who mostly work at night. As a remedy, we propose to use the *weighted amortized fairness* defined as below.

Definition 5 (Weighted Amortized Fairness). *The weighted amortized fairness of driver w is his/her accumulative earnings over his/her weighted active time:*

$$F_w = \frac{\sum_{t \in T} u_w^{(t)}/\xi^{(t)}}{\sum_{t \in T} a_w^{(t)}} \tag{3}$$

where $\xi^{(t)}$ is the weight associated to the earnings to normalize the variations in potential earnings across different times of the day. Typically, the time horizon of interest T is a *day*, a batch t is at the scale of *seconds e.g.*, 2 seconds, and the temporal weight $\xi^{(t)}$ varies on an *hourly* basis. This is because reports show that taxi drivers concern about hourly and daily wages for decision-makings [16] and research have demonstrated dramatic fluctuations in taxi demand by hours [20]. For simplicity, we use the median of driver earnings within the current hour containing batch t as $\xi^{(t)}$.

To quantify the *fairness distribution among all drivers*, we define the *temporal earnings fairness* based on the weight amortized fairness as follows.

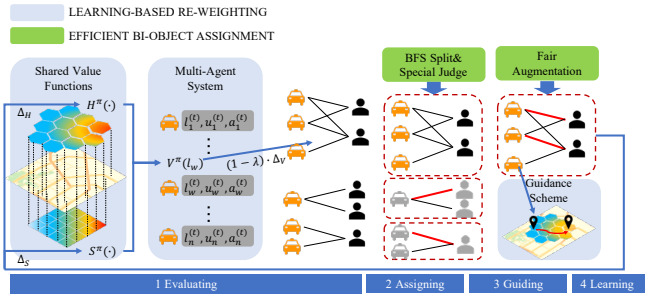


Figure 2: Workflow of LAF. In each batch, the learning-based re-weighting module first refines the edge weight of the given bipartite graph considering temporal dependencies across assignments. Then the efficient bi-objective assignment module finds an assignment via fair augmentation and other acceleration techniques. Lastly, the learning-based re-weighting module updates the weights based on the assignment result and guides idle drivers if necessary.

Definition 6 (Temporal earnings Fairness). *Given the reusable set of drivers W and the dynamically appeared set of requests R, the temporal earnings fairness among W is measured by an entropy variant of weighted amortized fairness:*

$$F = -\sum_{w \in W} \log\left(\frac{F_w}{\max_{w' \in W} F_{w'}}\right) \tag{4}$$

A large F indicates dispersed weighted amortized fairness across drivers, *i.e.*,, high earnings inequality. Thus we aim to minimize F.

Table 1 summarizes the frequently used notations in this paper.

Remarks. It is challenging to optimize the *total utility* U and the *temporal earnings fairness* F analytically for large-scale ride hailing applications. Prior solutions [13, 17] have the following limitations.

- Ignorance of the temporal dependency between *current* and *future* assignments. Many existing proposals oversimplified the problem setting by assuming independent arrival of new drivers in the time horizon. However, the drivers' availability and locations in the next batch are influenced by the assignments in current batch. Such dependency affects the optimization of utility and fairness, particularly at short-term scale where taxi demand and supply may fluctuate dramatically in space and time.
- Inefficient for *large-scale* applications. Previous research resorts to linear programming to solve the bi-objective matching problem, which can be slow to achieve real-time response to large-scale ride hailing.

4 METHOD

In this section, we introduce Learning to Assign with Fairness (LAF), a novel solution to optimize both the total utility and temporal earnings fairness for large-scale ride hailing applications. LAF adopts a *reinforcement learning based re-weighting* scheme to explicitly account for the temporal dependency between current and future assignments during matching. The scheme is implemented in an online manner to adapt to the dynamics of taxi supply and demand. LAF also incorporates a set of pruning and acceleration strategies for *efficient bi-objective (utility and fairness) assignment* on large-scale data. We present an overview of LAF in Sec. 4.1 and elaborates on the details in Sec. 4.2 and Sec. 4.3.

4.1 LAF Overview

LAF consists of a *learning-based re-weighting* module and an *efficient bi-objective assignment* module (Fig. 2).

- The *learning-based re-weighting* module (Sec. 4.2) refines the edge weights in the bipartite graph, which are initialized as trip prices (see Definition 3), to reflect *the impact of the current assignment on future utility and fairness.* The weight refinement policies are acquired via *online reinforcement learning* and we also design a *driver guidance scheme* to mitigate the cold start problem of online learning.
- The *efficient bi-objective assignment* module (Sec. 4.3) assigns taxi requests to drivers considering both utility and fairness. Its core is the *fair augmentation* algorithm, which applies the Kuhn–Munkres algorithm [8] to maximize utility and checks earnings rates across drivers to ensure fairness. We also design techniques (*BFS Split* and *Special Judge*) to accelerate the assignment on large-scale data taking advantage of the sparsity in the bipartite graph.

In each batch, LAF operates in four stages: evaluating, assigning, guiding and learning (Fig. 2). Each batch starts with evaluating. Given a bipartite graph with edge weights initialized by the trip price, the learning-based re-weighting module will update the weights such that the weights reflect both the immediate and future earnings. Then in the assigning stage, the efficient bi-objective assignment module computes a new matching on the refined bipartite graph, considering both the utility and fairness. Since *(i)* the updated weights account for the impact of the current assignments on future ones, and *(ii)* the assignment algorithm is bi-objective, the resulting assignments optimize both utility and fairness in a holistic manner (*i.e.*, considering temporal dependencies among assignments across batches). The last are the learning and guiding stages. The re-weighting module will learn from the matching results to improve its re-weighting policy for the next batch and guide idle drivers to busy area to avoid the cold start of online learning.

4.2 Learning-based Re-weighting

This subsection explains how to apply online reinforcement learning to model the impact of current assignments on future utility and fairness. We start with the basic formulation (Sec. 4.2.1), discuss practical issues (Sec. 4.2.2, Sec. 4.2.3, Sec. 4.2.4), and finally present the complete design (Sec. 4.2.5).

4.2.1 Online Reinforcement Learning based Formulation. Reinforcement learning [18] is a learning method by agents interacting with the environment over time. The agent takes an action each step and gets rewards from the environment. Based on the rewards, the agent updates its value function V_w^π which maps from the agent w's state to the expected accumulated reward if the agent takes action following policy π. The optimal policy can be learned through evaluating the rewards obtained through interactions with the environment.

In our context, each active driver is considered as an agent. In each batch, an agent (*i.e.*, driver) can take two actions, taking a taxi request r or remaining idle and the reward is p_r or 0, respectively. The state of a driver w is represented by his/her location $l_w^{(t)}$ and batch t as $(l_w^{(t)}, t)$. We are interested in the value function $V_w^\pi(l_w^{(t)}, t)$ of driver w for his/her state $(l_w^{(t)}, t)$ following the policy π that optimizes the total utility and temporal earnings fairness. Note that it is challenging to explicitly derive the policy π due to the fairness goal and the potential action conflicts among drivers. Hence we use value-iteration to learn the value function. That is, we iteratively update the value function from the rewards without considering policy as follows.

$$V_w^\pi(l_w^{(t)}, t) \leftarrow V_w^\pi(l_w^{(t)}, t) + \beta \cdot \Delta_w \ (\forall w \in W) \tag{5}$$

where β is learning ratio and Δ_w is decided by:

$$\Delta_w = \begin{cases} p_r + V_w^\pi(d_r, t+\tau_r) - V_w^\pi(l_w, t) & if\ w\ gets\ r \\ 0 & if\ w\ is\ idle \end{cases} \tag{6}$$

Since we do not aim for an explicit policy π, we omit the superscript π in the rest of paper for brevity. Also note that we adopt an online reinforcement learning model to be adaptive to the short-term dynamics in urban traffic. Combining traffic pattern prediction with reinforcement learning is out of the scope of this paper.

4.2.2 Reducing Numbers of States. The formulation above has too many states for an agent to explore, which prohibits effective reinforcement learning. Simple state discretization is insufficient. Note that the total number of states is the number of spacial states N_S times the number of temporal states N_T. Consider a city partitioned into tiles with a square of 1 km^2 and the time horizon of one day into segments with span of 20 min (average trip duration), then N_S and N_T will be about 8,000 and 72, respectively, resulting in over 500,000 states in total. This is beyond an agent's exploration in a day of 25,200 actions, estimated by assuming active time of 14 hours (permitted maximum working hours) and exploring a state every 2 seconds (a batch). The actual exploration number is much lower if excluding the time when serving requests.

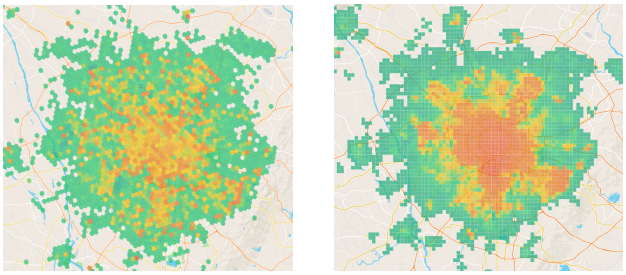
We reduce the number of states by two methods.

- *Spatial value function approximation.* We approximate the original value function in the spatiotemporal state space into the spatial state space only, *i.e.*, $V_w(d_r, t+\tau_r) = V_w(d_r, t)$. This is reasonable because most requests last less than half an hour and the variation of value functions can be ignored. Accordingly, the update equation can be rewritten as:

$$\Delta_w = \begin{cases} p_r + \gamma^{\tau_r} V_w(d_r) - V_w(l_w) & if\ w\ gets\ r \\ 0 & if\ w\ is\ idle \end{cases} \tag{7}$$

  The discount factor γ corrects the approximation inaccuracy of long-duration requests.
- *Information sharing among agents.* We use a shared value function combining all agents' value functions. This is reasonable because drivers in similar spatiotemporal state should have a similar evaluation to the location. Accordingly, the update equation is further reduced into:

$$V(l) \leftarrow V(l) + \beta' \cdot \sum_{w: l_w^{(t)} \in l} \Delta_w \tag{8}$$



(a) (b)

Figure 3: Hexagon and square value functions H and S at 17:00. Warmer colors indicate higher values.

where β' is the normalized learning ratio and l represents all possible locations of value functions.

The first method reduces the total number of states from $N_T \cdot N_S$ to N_S. The second method makes all drivers explore on the same value function to extend the exploration. Since the number drivers can be on the order of 10,000 in cities, each state will be explored a hundred times on average, which is enough for convergence.

4.2.3 Adaptive to Different Urban Layouts. In practical ride hailing applications, the space is usually discretized [19, 25], which restricts how the value function can vary on the urban layouts. To eliminate such restrictions, we smooth the value function from different directions, which involves two steps: *(i)* discretize the locations into a two-layer structure and *(ii)* smooth different layer values.

In LAF, the two layers are the *hexagon layer* and the *square layer* where the city is split into hexagons and squares, respectively. The two shapes provide different specialities for smoothing. Hexagons have more directions and smoothing via these directions benefit irregular urban layouts. As for squares, the boundaries are parallel to longitude and latitude, which are fit for regular area. As shown in Fig. 3, the hexagon layer shows a radial pattern align with the shape of trunk roads (Fig. 3(a)), while the square layer shows a regional pattern indicating some busy areas (Fig. 3(b)).

Finally, we smooth the value function as follows:

$$V(l) = \frac{1}{|DIR_H| + |DIR_S|} \left(\sum_{x \in DIR_H} H(l+x) + \sum_{x \in DIR_S} S(l+x) \right) \tag{9}$$

where H and S are the value functions for hexagons and squares, which are updated by Eq. (8). DIR_H and DIR_S specify the smoothing directed offsets for the two layers.

4.2.4 Avoiding Cold Start in Online Learning. Due to the online learning manner, the value function is initialized as zero, causing the value evaluation to degrade to the initial weight (trip price). Such cold starts prohibit the assignments in initial batches to be future-aware. In response, we propose to guide drivers to appropriate areas in advance. The appropriate areas are decided by both the distance and increment of value function (*i.e.*, line 4 in Algorithm 3). For simplicity, we use hexagon areas (A_h) in value function H as the candidate guiding destinations.

Algorithm 1: Evaluating stage of re-weighting module.

Input: The bipartite graph $G^{(t)} =< W^{(t)}, R^{(t)}, E^{(t)} >$, the rejection probability λ, hexagon value function H, square value function S

Output: The bipartite graph after re-weighting $G^{(t)} =< W^{(t)}, R^{(t)}, E'^{(t)} >$

$E'^{(t)} \leftarrow \emptyset$
for $w, r, \theta_{w,r} \in E^{(t)}$ **do**
  calculate $V(l_w)$ and $V(d_r)$ by Eq. (9)
  $\theta_{w,r} \leftarrow (1 - \lambda_{w,r}) \cdot (p_r + \gamma^{\tau_r} V(d_r) - V(l_w))$
  $E'^{(t)} \leftarrow E'^{(t)} \bigcup (w, r, \theta_{w,r})$
end
return $E'^{(t)}$

Algorithm 2: Learning stage of re-weighting module.

Input: Assignment results $M^{(t)}$, drivers $W^{(t)}$, hexagon value function H, square value function S

Output: Value functions H and S after update

for w *in* $W^{(t)}$ **do**
  if *there exists an order* r, *s.t.* $(w, r) \in M^{(t)}$ **then**
    $\Delta_H \leftarrow p_r + \gamma^{\tau_r} H(d_r) - H(l_w)$
    $\Delta_S \leftarrow p_r + \gamma^{\tau_r} S(d_r) - S(l_w)$
  end
  else
    $\Delta_H \leftarrow 0, \Delta_S \leftarrow 0$
  end
  $H \leftarrow H + \beta \cdot \Delta_H, S \leftarrow S + \beta \cdot \Delta_S$
end
return H, S

4.2.5 Putting It Together. The learning-based re-weighting module consists of three stages: evaluating, learning, and guiding. Algorithm 1,2 and 3 illustrate the processes, respectively. In line 4 of Algorithm 3, Function *dist* calculates the distance.

Note that the re-weighting module exploits reinforcement learning to explicitly consider the impact of current assignments on future ones. It does not directly contribute to fairness. In LAF, optimizing the fairness objective is embedded into the optimization of the total utility, as we will explain next.

4.3 Efficient Bi-Objective Assignment

This subsection presents our assignment algorithms that optimize both utility and fairness.

4.3.1 Fair Augmentation. Unlike previous bi-objective solutions that either rely on the slow linear programming [17] or conduct repetitive reassignments between separate optimizations on fairness and utility [13], we propose to directly embed fairness check into the process to maximize the utility for faster assignments.

The standard method to find a matching on a bipartite graph that maximizes utility is the Kuhn-Munkras algorithm [8], where the basic operation is augmentation. That is, we firstly try to find a path composed of matched and unmatched edge alternatively,

Algorithm 3: Guiding stage of re-weighting module.

Input: Idle drivers $W_i^{(t)}$, candidate guiding destinations A_h, weight ξ from batch 1 to current batch t

Output: Guidance Scheme $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$
Sort drivers based on $\frac{\sum_{i=1}^{t} u_w^{(i)} / \xi^{(i)}}{\sum_{i=1}^{t} a_w^{(i)}}$ ascendingly.
for $w \in W^{(t)}$ **do**
  $g \leftarrow \arg\max_{g' \in A_h} \frac{V(g') - V(l_w)}{dist(g', l_w)}$
  $\mathcal{S} \leftarrow \mathcal{S} \bigcup (w, g)$
end
return $\mathcal{S}$

while the summation of the unmatched edges' weights is larger than that of matched edges, and if found, we switch the unmatched and matched edges to increase the total utility. The idea of our fair augmentation algorithm is, when searching for an augmentation path, we check the drivers' future earnings ratios and reject the augmentation with large earnings ratios variance. However, rejecting augmentation may impair the achieved utility. Consider the situation where a newly online driver in the augment path, because he/she has almost zero idle time, so his/her earnings ratio after serving a request will be very large, which leads to frequent rejection of augmentation. Also note that checking the variance of all drivers after finding an augment path is time-consuming. Therefore, instead of checking all drivers, we only check the adjacent two drivers in the augment path, and once we find variance ratio, we break and stop augmentation immediately. Algorithm 4 illustrates the fair augmentation algorithm, which has a time complexity of $O(N^2M)$, where $M = \max(|W^{(t)}|, |R^{(t)}|)^2$ and $N = \min(|W^{(t)}|, |R^{(t)}|)$.

4.3.2 Accelerations. We further accelerate the fair augmentation algorithm taking advantage of the sparsity of the bipartite graph. Specifically, we perform breadth first search (BFS) to split the assignment graph into several parts. Many of these parts contain only one driver (request), so we use special judge to quickly find the best request (driver) for the single node. Our bi-objective assignment algorithm with further accelerations is shown in Algorithm 5.

5 EVALUATION

This section presents the evaluations of our methods.

5.1 Experiment Setting

Validation Environment. We conduct experiments on a simulator based on a major ride hailing platform. We experiment on three cities for 21 days. The simulator generates requests, simulates the drivers and passengers' behaviors (*i.e.*, idle driver transitions and passenger rejections), builds the bipartite graph, and executes our assigning algorithm.

Baselines. We compare our method with the following baselines.

- Distance-Greedy (DG): it assigns every request to its closest available driver. It is a widely used baseline in ride hailing.
- Earnings-Ratio-Greedy (ERG): it first sorts all available drivers by weighted amortized fairness F_w in a descending order

Algorithm 4: Fair augmentation.

Input: The bipartite graph $G^{(t)} =< W^{(t)}, R^{(t)}, E^{(t)} >$, current batch t, weight ξ from batch 1 to t
Output: Assignment $M^{(t)}$

$M^{(t)} \leftarrow \emptyset$
for $w \in W^{(t)}$ **do**
  while *there exists an augment Path P* **do**
    $accept \leftarrow True$
    for *every pair of adjacent drivers* w_{pre}, w_{cur} **do**
      $r_{pre} \leftarrow$ *current assigned request of* w_{pre}
      $r_{cur} \leftarrow$ *current assigned request of* w_{cur}
      if $|\frac{\sum_{i=1}^{t} \frac{u_{w_{pre}}^{(i)}}{\xi^{(i)}} + \frac{pr_{pre}}{\xi^{(t)}}}{\sum_{i=1}^{t} a_{w_{pre}}^{(i)} + \tau_{r_{pre}}} - \frac{\sum_{i=1}^{t} \frac{u_{w_{cur}}^{(i)}}{\xi^{(i)}} + \frac{pr_{cur}}{\xi^{(t)}}}{\sum_{i=1}^{t} a_{w_{cur}}^{(i)} + \tau_{r_{cur}}}| > \epsilon$ **then**
        $accept \leftarrow False$
        break
      end
    end
    if *accept* **then**
      update $M^{(t)}$ *based on P.*
    end
  end
end
return $M^{(t)}$

Algorithm 5: Assignment algorithm.

Input: The bipartite graph $G^{(t)} =< W^{(t)}, R^{(t)}, E^{(t)} >$
Output: Assignment $M^{(t)}$

$P^{(t)} \leftarrow$ split $G^{(t)}$ by BFS.
$M^{(t)} \leftarrow \emptyset$
for $p \in P^{(t)}$ **do**
  if *there is only one driver w (request r)* **then**
    $M_p \leftarrow$ <w(best driver), best request(r)>
  end
  else
    $M_p \leftarrow$ FairAugmentation(p)
  end
  $M^{(t)} \leftarrow M^{(t)} \bigcup M_p$
end
return $M^{(t)}$

and sorts requests by rewards per batch(p_r/τ_r) in a descending order. Then it assigns request to drivers respectively based on the order.

- Integer-Linear-Programming (ILP) [17]: it is the first work that applies amortized fairness to assess the earnings fairness in ride hailing. It converts the problem into an integer linear programming problem. For fair comparison, we omit the passenger-side fairness from its optimization objective.
- Reassign (REA) [13]: it first calculates two assignments optimizing utility and fairness (adjust to $\min_w F_w$ for equal comparison), respectively. Then it reassigns matching from one assignment to the other to find a trade-off between utility and fairness. This is the state-of-the-art solution that optimizes both utility and fairness in ride hailing.

Table 2: Overall results on fairness F and utility U of different assignment algorithms. A smaller F means better fairness and a larger U means higher utility.

City	Method	F (weekday)	U (weekday)	F (weekend)	U (weekend)
A	DG	41,767	2,242,977	31,962	2,131,996
	ERG	48,549	2,256,277	35,903	2,172,489
	ILP [17]	44,072	2,246,077	30,573	2,122,834
	REA [13]	44,251	2,250,118	27,865	2,134,534
	LAF (ours)	**22,656**	**2,656,773**	**13,384**	**2,565,060**
B	DG	31,217	1,297,735	28,563	1,360,659
	ERG	38,594	1,373,478	37,464	1,452,488
	ILP [17]	29,496	1,281,273	25,077	1,336,712
	REA [13]	31,744	1,319,958	33,671	1,373,691
	LAF (ours)	**7,420**	**1,479,348**	**4,976**	**1,616,385**
C	DG	12,897	872,693	9,044	865,031
	ERG	13,704	885,558	8,793	882,573
	ILP [17]	12,377	862,588	8,439	863,245
	REA [13]	13,491	880,847	8,684	881,519
	LAF (ours)	**2,553**	**1,114,000**	**2,392**	**1,109,474**

Evaluation Metrics. We assess the performance of different methods by *fairness*, *utility* and *efficiency*, where fairness is measured by the temporal earnings fairness (Definition 6), utility by the total utility (Definition4) and efficiency by hourly accumulated consuming time for assignment.

Implementation. All the algorithms are implemented in Python3. We set the discount factor γ (in Eq. (7)) to 0.9 and learning rate β' (in Eq. (8)) to 0.025. The experiments are conducted on Intel Xeon CPU E5-2630 v4 @ 2.20GHz with 12GB memory.

5.2 Overall Performance

Table 2 summarizes the fairness and utility metrics of different assignment methods on datasets of the three cities. Overall, LAF improves the fairness metric by 45.7% ~81.4% than the baselines on weekdays and 52.0% ~86.7% on weekends. LAF also achieves the highest total utility, which is 7.7% ~29.1% higher than the baselines on weekdays and 11.3% ~28.5% on weekends. In particular, our LAF outperforms ILP by 68.4% and 22.1% on average in terms of fairness and utility, respectively. As for REA, LAF gains an average improvement of 69.3% and 20.1% in terms of fairness and utility, respectively. Also, LAF consistently outperforms the baselines in all the three cities.

Fig. 4 compares the efficiency of different algorithms. The simple baselines DG and ERG have a time complexity of $O(|E| \log |E|)$ and $O(|W| \log |W|)$, respectively. They are fast but perform poorly in terms of fairness and utility *e.g.*, 114.6% and 14.2% worse than LAF in fairness and utility at weekends in City A. Our LAF is almost as fast as these two simple solutions, and the running time of LAF is relatively stable across all hours of the day. ILP and REA can be up to 797% slower than LAF. Also their running time is sensitive to the variations in traffic. During the rush hours when the number of requests surges rapidly, the time cost of these two methods also increase significantly.

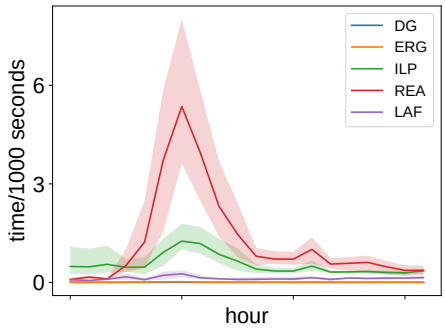


Figure 4: Execution time of each algorithm to hours in a day on City A.

5.3 Experimental Result Analysis

We now take a deeper look at the performance of different methods on datasets of City A, the largest in scale among the three cities, to further understand the effectiveness of our method.

5.3.1 Why LAF Outperforms Others: A Case Study. As an illustration on the difference between different assignment algorithms, we plot the trajectories of a driver in Fig. 5.

- DG assigns the driver the nearest request regardless of fairness. After the second assignment, the driver is trapped in a remote area and cannot go back for other request (Fig. 5(a)).
- ERG assigns the driver with the lowest earnings ratio with the request of the highest p_r/τ_r. After the second request, the driver suffers a low earnings ratio and is assigned a long-distance request (Fig. 5(b)).
- ILP makes the same assignment as DG for this driver (Fig. 5(c)). This is because after the driver serves a long-distance request, his/her earnings ratio is relatively high, resulting in a short-distance request for the next batch. Then in a remote area, the relatively high earnings ratio after serving the two requests makes it difficult for the driver to get another request.
- REA accounts for both fairness and utility. The optimization for utility increases the number of requests assigned to the driver (Fig. 5(d)). But still, REA is unaware of the destination and the driver can only pick a short-distance request and may be trapped in a remote area.
- Our LAF makes assignments by considering their impact on the future assignments. The guidance also avoids the driver being trapped in remote areas. As shown in Fig. 5(e), after serving some requests in the busy area, the driver has a relatively high earnings ratios. Then the driver is assigned requests with remote destinations to be fair. Afterwards, to avoid being idle for too long, the driver is guided to areas where he/she can get requests and return to the busy area.

5.3.2 Correlation between Traffic Dynamics and Fairness. One of our contributions is to use the *weighted* amortized fairness F_w as the fairness metric (Definition 5). We justify the advantages of our weighted amortized fairness metric by showing the correlation between the weight $\xi^{-1}(t)$ in F_w and the traffic dynamics.

Fig. 6(a) plots the numbers of drivers and requests as well as the weight $\xi^{-1}(t)$ over different hours of a day. The numbers of drivers and requests reflect the variations of traffic. As is shown, the traffic goes up from 6 a.m. to 8 a.m., fluctuates during the daytime, and drops at night. The weight $\xi^{-1}(t)$ shows the same trend. It indicates we put more weight of fairness during rush hours, where there are more drivers and thus requiring more fairness. Also note that the median of driver earnings within the current hour as ξ is a good option because it changes slightly earlier than the traffic. This is because experienced drivers tend to turn active a bit earlier before the rush hours, causing the median of earnings drops earlier and thus the weight ξ increases predictably.

Fig. 6(b) shows the *hourly* earnings of 3,500 randomly sampled drivers. As is shown, the distribution of drivers' hourly earnings are relatively even when using our fairness metric as an optimization objective, and our LAF algorithm performs the best in reducing the number of drivers with extremely high/low hourly earnings.

Table 3: Comparison of *unweighted* amortized fairness (used in [17] and [13]) and utility. DG is excluded because it does not no fairness. The utility results of the algorithms remain the same as Table 2.

Method	F (weekday)	U (weekday)	F (weekend)	U(weekend)
DG	44, 442	2, 242, 977	33, 057	2, 131, 996
ERG	54, 270	2, 248, 669	38, 315	2, 166, 980
ILP[17]	40, 066	2, 243, 334	30, 772	2, 120, 947
REA[13]	42, 003	2, 242, 561	31, 832	2, 130, 431
LAF (ours)	**20, 539**	**2, 359, 431**	**12, 963**	**2, 447, 596**

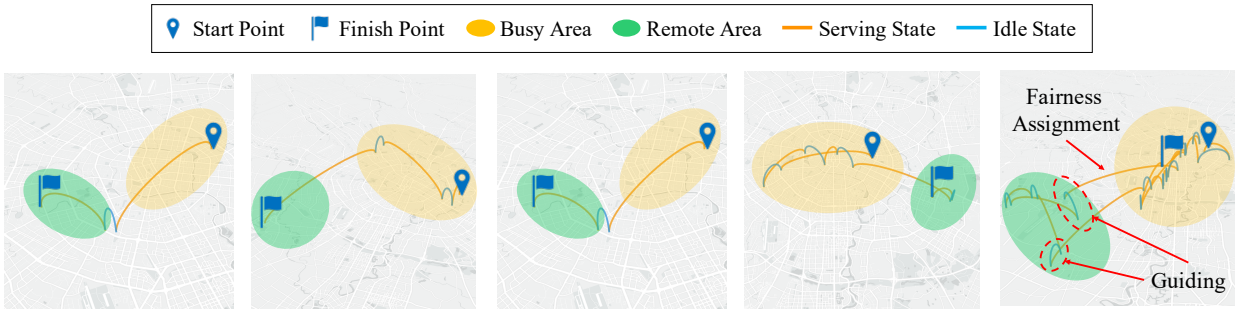
5.3.3 Effectiveness in Optimizing Unweighted Amortized Fairness. Table 3 compares different algorithms on optimizing the *unweighted* amortized fairness (*i.e.*, set $\xi^{(t)}$ to 1 for all t) and utility. The results shows that our LAF still achieves 57.9% ~62.2% improvement in fairness and 17.3% ~17.6% improvement in utility, even though ILP and REA are designed to optimize the unweighted amortized fairness. Therefore, our LAF outperforms the state-of-the-arts in both unweighted and weighted amortized fairness.

6 CONCLUSION

In this paper, we propose *Learning to Assign with Fairness* (LAF) that effectively and efficiently optimizes the total utility (expected total earnings of drivers) and driver fairness (weighted amortized fairness of driver earnings). The key novelty is to apply reinforcement learning to make assignments that explicitly account for the dependency among assignments such that both utility and fairness can be optimized in a holistic view. LAF also incorporates a set of techniques to stay adaptive to traffic dynamics and different urban layouts, and make fast assignments over large-scale of data. Experimental results show that LAF achieves 86.7%, 29.1% and 797% improvement in terms of fairness, utility and efficiency over the state-of-the-art fair assignment algorithms for ride hailing. We envision our work as a guideline for practical adoption of fair task assignment in real-world ride hailing applications.

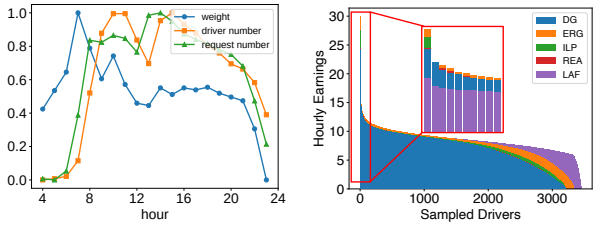
ACKNOWLEDGMENTS

We are grateful to anonymous reviewers for their constructive comments. This work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC)



(a) DG (b) ERG (c) ILP (d) REA (e) LAF

Figure 5: A driver's trajectories under different assignment algorithms.



(a) (b)

Figure 6: (a) illustrates min-max normalization of $\xi^{-1}(t)$, driver number and request number in different hours of a day. (b) illustrates systematic sampled 3.5k drivers sorted in descending order of hourly earnings.

under Grant Nos. 61822201, U1811463 and 62076017, the Hong Kong RGC TRS T41-603/20-R, the CAAI Huawei MindSpore Open Fund No. CAAIXSJLJJ-2020-020-A, and the State Key Laboratory of Software Development Environment Open Funding No. SKLSDE-2020ZX-07.

REFERENCES

[1] Barbara M. Anthony and Christine Chung. 2014. Online bottleneck matching. *Journal of Combinatorial Optimization* 27, 1 (2014), 100–114.
[2] Eszter Bokányi and Anikó Hannák. 2020. Understanding inequalities in ride-hailing services through simulations. *Scientific reports* 10, 1 (2020), 1–11.
[3] Ria Mae Borromeo, Thomas Laurent, Motomichi Toyama, and Sihem Amer-Yahia. 2017. Fairness and Transparency in Crowdsourcing. In *Proc. EDBT*. OpenProceedings.org, Venice, Italy, 466–469.
[4] Niv Buchbinder and Joseph (Seffi) Naor. 2006. Fair Online Load Balancing. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures*. ACM, New York, NY, USA, 291–298.
[5] Zhao Chen, Peng Cheng, Lei Chen, Xuemin Lin, and Cyrus Shahabi. 2020. Fair Task Assignment in Spatial Crowdsourcing. *Proceedings of the VLDB Endowment* 13, 11 (2020), 2479–2492.
[6] Yuga J. Cohler, John K. Lai, David C. Parkes, and Ariel D. Procaccia. 2011. Optimal Envy-Free Cake Cutting. In *Proc. AAAI*. AAAI, San Francisco, CA, USA, 626–631.
[7] John P. Dickerson, Ariel D. Procaccia, and Tuomas Sandholm. 2014. Price of fairness in kidney exchange. In *Proc. AAMAS*. Springer, Paris, France, 1013–1020.
[8] Jack Edmonds and Richard M Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19, 2 (1972), 248–264.
[9] Robert S Garfinkel. 1971. An Improved Algorithm for the Bottleneck Assignment Problem. *Operations Research* 19.7 (1971), 1747–1751.
[10] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proc. NSDI*. USENIX Association, Boston, MA, USA, 323–336.
[11] David R. Karger, Sewoong Oh, and Devavrat Shah. 2014. Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems. *Operations Research* 62, 1 (2014), 1–24.
[12] Ian A. Kash, Ariel D. Procaccia, and Nisarg Shah. 2014. No Agent Left Behind: Dynamic Fair Division of Multiple Resources. *Journal of Artificial Intelligence Research* 51 (2014), 579–603.
[13] Nixie S. Lesmana, Xuan Zhang, and Xiaohui Bei. 2019. Balancing Efficiency and Fairness in On-Demand Ridesourcing. In *Proc. NeurIPS*. Curran Associates Inc., Vancouver, BC, Canada, 5310–5320.
[14] Max O. Lorenz. 1905. Methods of measuring the concentration of wealth. *Publications of the American statistical association* 9, 70 (1905), 209–219.
[15] Duncan C. McElfresh and John P. Dickerson. 2018. Balancing Lexicographic Fairness and a Utilitarian Objective With Application to Kidney Exchange. In *Proc. AAAI*. AAAI, New Orleans, LA, USA, 1161–1168.
[16] Hilary C Robinson. 2017. *Making a digital working class: Uber drivers in Boston, 2016-2017*. Ph.D. Dissertation. Massachusetts Institute of Technology.
[17] Tom Sühr, Asia J. Biega, Meike Zehlike, Krishna P. Gummadi, and Abhijnan Chakraborty. 2019. Two-Sided Fairness for Repeated Matchings in Two-Sided Markets: A Case Study of a Ride-Hailing Platform. In *Proc. KDD*. ACM, Anchorage, AK, USA, 3082–3092.
[18] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press, Cambridge.
[19] Xiaocheng Tang, Zhiwei (Tony) Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. 2019. A Deep Value-network Based Approach for Multi-Driver Order Dispatching. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*. ACM, Anchorage, AK, USA, 1780–1790.
[20] Yongxin Tong, Yuqiang Chen, Zimu Zhou, Lei Chen, Jie Wang, Qiang Yang, Jieping Ye, and Weifeng Lv. 2017. The Simpler The Better: A Unified Approach to Predicting Original Taxi Demands based on Large-Scale Online Platforms. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*. ACM, Halifax, NS, Canada, 1653–1662.
[21] 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. *PVLDB* 9, 12 (2016), 1053–1064.
[22] Xing Wang, Niels Agatz, and Alan Erera. 2018. Stable matching for dynamic ride-sharing systems. *Transportation Science* 52, 4 (2018), 850–867.
[23] Yansheng Wang, Yongxin Tong, Cheng Long, Pan Xu, Ke Xu, and Weifeng Lv. 2019. Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach. In *Proc. ICDE*. IEEE Press, Macao, China, 1478–1489.
[24] Pan Xu, Yexuan Shi, Hao Cheng, John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, Yongxin Tong, and Leonidas Tsepenekas. 2019. A Unified Approach to Online Matching with Conflict-Aware Constraints. In *Proc. AAAI*. AAAI, Honolulu, HI, USA, 2221–2228.
[25] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. 2018. Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*. ACM, London, UK, 905–913.
[26] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. 2017. A Taxi Order Dispatch Model based On Combinatorial Optimization. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*. ACM, Halifax, NS, Canada, 2151–2159.