Utility-Aware Dynamic Ridesharing in Spatial Crowdsourcing

Yafei Li, Huiling Li, Xin Huang, Jianliang Xu, Yu Han, and Mingliang Xu

Abstract—With smartphones and geo-locating devices widely used around the world, ridesharing, as a main application field of spatial crowdsourcing, has been fast expanding its widespread adoption and potentially brings great benefits to human society and the environment. However, ridesharing has so far not been as popular as expected. A recent survey shows that notable obstacles come from concerns about social comfort and price fairness when riding with strangers. To defeat these obstacles, in this paper, we study a novel problem of utility-aware ride matching (URM) for dynamic ridesharing, where drivers and riders are matched in batches by considering their social comfort and price revenue. While the URM problem is of practical usefulness, we prove that this problem is Nondeterministic Polynomial Hard (NP-hard). To tackle the problem optimally and find exact answers, we present a novel bipartite matching algorithm by integrating an effective Driver-Rider Graph (DR-Graph) index. To balance the effectiveness and efficiency, we propose two efficient algorithms to solve the URM problem with only a small loss of utility. Leveraging a bounded Driver-Rider-group Graph (DRg-Graph) and several useful pruning bounds for matching utility and travel cost, we develop an ϵ-refining algorithm to find an ϵ-approximation matching utility of the optimal answer. Extensive experiments on real datasets showed that our proposed algorithms achieved effective matching results of social-pricing based utilities and the efficient performance of quick matching under various parameter settings.

Index Terms—Spatial crowdsourcing, ridesharing, social network, location-based services, optimization.

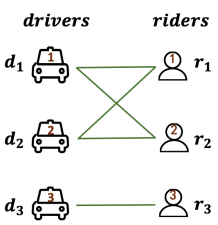
1 Introduction

NOWADAYS, smartphones, equipped with rich built-in sensors (e.g., GPS and accelerometers) and multiple radios (e.g., Wi-Fi and cellular), become ubiquitous and are widely used around the world. This trend leads to a new computing paradigm, spatial crowdsourcing (SC), that can be fully explored for real-time and location-aware crowdsourced tasks. As a main application field of SC, ridesharing, which emerged as the times require, has greatly facilitated people's daily commutes. Major commercial platforms, such as DiDi [1], Uber [4], and Lyft [2], offer ridesharing services that enable multiple riders to travel towards the same direction to share rides with drivers. So far, however, ridesharing has not been becoming as popular as expected, e.g., only 35 percent of trips are shared nationwide in the United States [3], which eliminates many professed benefits, such as reduced congestion and carbon output. As reported in a recent survey [24], notable obstacles in current ridesharing are the lower utility about social comfort and pricing fairness, which affects the customer satisfaction and quality of experience (QoE), the keys to the platform's retention of customers and the enthusiasm for encouraging customers to participate in ridesharing [21], [22], [33]. Thus, in this paper, we focus on the utility-aware ridesharing taking both social comfort and pricing fairness into account.

- *Y. Li, H. Li, Y. Han and M. Xu are with the School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. E-mail: {ieyfli, iexumingliang}@zzu.edu.cn, {huilingli, yhan}@gs.zzu.edu.cn. Corresponding authors: M. Xu and J. Xu*
- *X. Huang and J. Xu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong SAR, China. E-mail: {xinhuang, xujl}@comp.hkbu.edu.hk.*

In the literature, there is a line of existing works that examine issues related to utility-aware carpooling [9], [23], [14], [15], [24], [17], [43], [10]. However, most of these works only focus on one aspect of social comfort and pricing fairness. On one hand, in terms of *social comfort*, Li *et al.* [23] and Cheng *et al.* [14] suggested enforcing a specific social constraint for ride group formulation. However, the social models adopted in [15] and [23] are too restrictive and thus reduce the ridesharing success rate. To address this weakness, the techniques presented in [24] and [17] make use of social distance to model social comfort among drivers and riders, in which the social constraint is relaxed to a certain extent. On the other hand, in terms of *pricing fairness*, current commercial platforms provide a *static* pricing scheme for riders by pre-calculating the sharing cost (i.e., by giving a fixed discount on the original travel cost even without passengers sharing a ride). Obviously, such a pricing scheme is unfair for long-trip riders, since they must bear more detours on their entire trip caused by picking up and dropping off other riders. Several recent works [9], [43], [10] have attempted to study the problem of dynamic pricing by proposing a system where the rider's fare is dynamic and proportional to the actual detour cost. It is certain that a dynamic pricing mechanism makes the rider's fare fairer and more flexible. To sum up, since social comfort and pricing fairness are two primary concerns that affect riders' willingness to participate in ridesharing, they should be offered simultaneously instead separately.

To accomplish this, we study a novel utility-aware ride matching (URM) problem for dynamic ridesharing. Given a set of drivers and a stream of riders, the URM-problem aims to find out the matching plan for drivers and riders that offers maximum platform utility in terms of social



Matching	Utility
(d_1, r_1)	24
(d_2, r_1)	16
(d_1, r_2)	21
(d_2, r_2)	20
(d_3, r_3)	18

Fig. 1: A toy example of the URM-problem.

comfort and price revenue. Figure 1 depicts a toy example for the URM-problem. Assume that there exist three riders r_1, r_2, r_3, and three drivers d_1, d_2, and d_3. The available number of seats with d_1, d_2, and d_3 are 1. The matching utilities for drivers and riders are listed in Figure 1. Here, the matching means that the driver can pick up and drop off the rider subject to his/her trip constraints. Therefore, there exist two candidate matching plans, including $\mathcal{P}_1 = \{(d_2, r_1), (d_1, r_2), (d_3, r_3)\}$ with utility 16+21+18=55 and $\mathcal{P}_2 = \{(d_1, r_1), (d_2, r_2), (d_3, r_3)\}$ with utility 24+20+18=62. Thus, $\mathcal{P}_2$ is the optimal matching plan.

The integrated consideration of social comfort and pricing revenue in the URM-problem brings non-trivial challenges. First, it is difficult to model the utility of drivers and riders acting under the influence of social comfort and price revenue. Second, the URM-problem is shown to be NP-hard in our theoretical analysis. Thus, a baseline exact method requires a time complexity of dynamically assigning drivers to serve more riders, which is exponential to the number of drivers and riders. To satisfy real-time matching requirements, we need to design a highly efficient algorithm. In this paper, we first present an integer linear programming algorithm to solve the URM-problem optimally. Then, we use a bipartite graph to model the URM-problem and present an efficient DR-Graph-based bipartite matching algorithm. To further accelerate the search speed, we present a fast matching algorithm and an approximate algorithm to solve the URM-problem with a quality guarantee. The major contributions of this paper are summarized as follows:

- We first present a framework of ridesharing matching, equipped with four entities of road networks, social networks, drivers, and riders. Based on the constraints of drivers' route schedules, we formally give a definition of matching utility based on the quantified metrics of social comfort and price revenue. Then, we define a novel URM-problem to find the best matching plan for drivers and riders with the maximum utility, and theoretically analyze the hardness of the URM-problem.
- We present two exact algorithms for the URM-problem to find optimal matching plans. We first reformulate the URM-problem and use an integer linear programming algorithm to solve it. Furthermore, we propose a novel bipartite matching algorithm by integrating an effective DR-Graph structure to find optimal answers.
- We propose two efficient algorithms, FMA and eRA, to balance the efficiency and effectiveness of solutions to the URM-problem. FMA is an efficient heuristic greedy algorithm that integrates an effective DRg-Graph index. Moreover, we design a bounded DRg-Graph and several useful pruning bounds for matching utility and travel cost. Integrating these pruning bounds, we develop an effective ϵ-refining algorithm eRA to find an ϵ-approximation matching utility of the optimal answer.
- We conduct extensive experiments on two real-world datasets to show that our proposed algorithms can achieve desirable performance under a wide range of parameter settings.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 formally defines the URM-problem. Section 4 proposes two exact algorithms to solve the URM-problem. Section 5 presents two efficient algorithms to find approximate matching answers. Extensive experiments are conducted in Section 6 to evaluate the performance of our proposed algorithms, followed by the conclusion and future work in Section 7.

2 Related Work

In this section, we survey the related studies on dynamic ridesharing and personalized ridesharing.

2.1 Dynamic Ridesharing

Dynamic ridesharing refers to matching drivers and riders on very short notice or even en-route. Unlike static ridesharing, dynamic ridesharing does not need pre-arranged commutes for drivers and riders, it plays an important role in ridesharing studies and attracts more attentions from both academia and the industry [37], [40], [42], [36], [31], [12], [18], [38].

Shuo et al. [27] developed a taxi-sharing system that accepts the real-time ride request and arranges appropriate taxis to serve them by ridesharing under different constraints. Cheng et al. [13] studied the maximum revenue vehicle dispatching problem and proposed efficient learning-based algorithms to predict vehicle demands. Ta et al. [35] studied the ridesharing problem from two approaches: search-based ridesharing and join-based ridesharing. They respectively proposed a best-first algorithm and an approximated algorithm to tackle the problem. To evaluate the feasibility of large-scale fleet operation, Alonso-Mora et al. [6] presented a real-time optimization request matching and dynamic vehicle route selection method for low-and-high-capacity vehicles. Liu et al. [25] considered the mobility-aware taxi ridesharing problem and presented a mT-Share method, which constructs an index on taxis and ride requests with both geographical information and travel directions to serve both online and offline ride requests. All the above works contribute significantly to ridesharing service but do not consider the ride experience to improve the arrangement quality for users. This differentiates them from the goal of our utility-aware ridesharing problem.

2.2 Personalized Ridesharing

Recently, personalized ridesharing has attracted lots of attention in the literature. In ridesharing studies, utility factors like price and social cohesion, are thought to improve the service quality [9], [39], [41], [32], [26].

To improve the pricing fairness, Chen et al. [10] presented a new price-aware ridesharing model. To accelerate

TABLE 1: Frequently used notations

Notation	Definition
$\mathcal{G}_n$, $\mathcal{G}_s$	a road network and a social network
r, d	a rider and a driver
$\mathcal{R}$, $\mathcal{D}$	a set of riders and a set of drivers
$\mathcal{V}_d$	the onboard riders of d
$\mathcal{R}_g$, $\mathcal{R}_G$	a rider group and a set of rider group
$\kappa(d, \mathcal{R}_g)$	the matching utility of d and $\mathcal{R}_g$
$\varphi(d, \mathcal{R}_g)$	the social comfort of d and $\mathcal{R}_g$
$\mu(d, \mathcal{R}_g)$	the price revenue for d serving $\mathcal{R}_g$
η	the benefit of unit travel cost
$\mathcal{P}$	a matching plan
$Score(\mathcal{P})$	the utility of $\mathcal{P}$
$\mathcal{G}_h$, $\mathcal{G}_b$	a DR-Graph and a bipartite graph
$\mathcal{G}_\mathcal{B}$,$\mathcal{BG}$	an DRg-Graph and a bounded DRg-Graph
$\kappa_l(d, \mathcal{R}_g)$	the matching utility lower bound of d and $\mathcal{R}_g$
$\kappa_u(d, \mathcal{R}_g)$	the matching utility upper bound of d and $\mathcal{R}_g$
$Score_{lb}(\mathcal{G}_B)$	the utility lower bound of $\mathcal{P}$ in $\mathcal{G}_B$
$Score_{ub}(\mathcal{G}_B)$	the utility upper bound of $\mathcal{P}$ in $\mathcal{G}_B$

the matching between drivers and riders, they designed two efficient matching algorithms that follow the single-side and dual-side searches. Pamula et al. [30] proposed a taxi-ridesharing service that reduces the total travelling distance per taxi and travelling cost per person. Wang et al. [28] proposed a new spatial query and answered the question of how to reduce travel cost. Li et al. [24] proposed a price-aware top-k problem in ridesharing, which used widespread ranking functions to rank the matching and developed several efficient algorithms. On the other hand, to further improve the ride experience, several other works have integrated social cohesion into ridesharing [23], [17], [15], [14], [7], [16], [34], [14]. Li et al. [23] proposed a social-aware ridesharing group formulation problem in which drivers and riders forming a ride group in terms of their social connections. To further enrich the ride experience, Fu et al. [17] considered both social cohesion and interest similarity to measure a ridesharing group and proposed several greedy algorithms to accelerate the matching efficiency. Cici et al. [15] studied some mobility datasets to understand mobility patterns and social ties among users, and then (by considering a range of trip constraints) proposed an efficient algorithm to match users with similar mobility patterns. Cheng et al. [14] opined that the riders' satisfaction is important, and then formulated the problem of utility-aware ridesharing to improve the ride experience. Despite those extensive studies on personalized ridesharing, the two primary concerns of social comfort and pricing fairness (which affect the willingness of passengers to participate in ridesharing) should be taken into account simultaneously instead of independently as they have been in these previous studies. In this work, we integrate both social comfort and pricing fairness in our consideration.

3 System Model and Problem Formulation

In this section, we first present the system model, then provide some preliminaries and the problem statement. Table 1 gives a summary of the notations used in this paper.

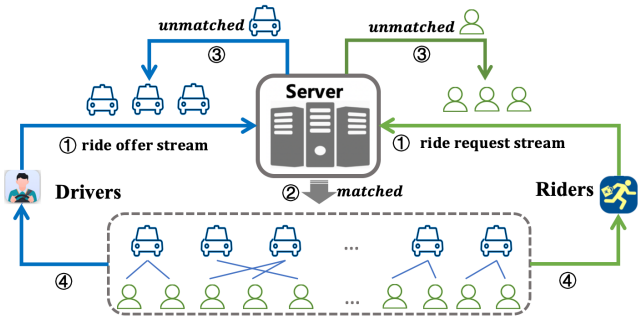


Fig. 2: The system model of ridesharing matching.

3.1 System Model

In a ridesharing service, there are many part-time drivers who are willing to offer available seats to the riders heading in the same direction as their own. Simply put, the travel pattern is that the driver starts from his/her origin to pick up the riders at their pick-up points, then takes the riders to their drop-off points, and finally ends the driver's trip at his/her own destination [24].

Figure 2 outlines our utility-aware ridesharing matching (URM) framework. The general processing framework consists of three parties, namely the drivers, the riders, and the server. Specifically, drivers submit *ride offers* and riders send in *ride requests* to the server in a stream fashion (e.g., Step-1); The server handles the arrived ride offers and ride requests in each batch by considering their overall utility in terms of social comfort and price revenue. Drivers and riders are respectively notified of their matched ride requests and ride offers (Step-2 and Step-4); The unmatched ride offers and ride requests are placed into the next batch for re-matching (Step-3). Note that the expired ride offers and ride requests are discarded from the server.

We clarify that our system is centralized. Once drivers enter the system, they should send their real-time status to the system periodically (e.g., every few seconds), including their current locations, travel routes, and the number of unoccupied seats. On the other hand, when a rider is assigned to a driver, his/her real-time status dynamically changes as the driver moves, including the rider's current location and actual pick-up and drop-off time.

3.2 Networks, Drivers, Riders, and Schedules

Our ridesharing matching problem consists of four entities, including a road network $\mathcal{G}_n$, a social network $\mathcal{G}_s$, drivers $\mathcal{D}$, and riders $\mathcal{R}$. We define them one by one as follows.

- Road network: denoted as $\mathcal{G}_n = (V_n, E_n)$, where V_n and E_n are the sets of road intersections and road segments, respectively. An edge $l_{ij} \in E_n$ is a road segment linking two road intersections l_i and l_j ($l_i, l_j \in V_n$).
- Social network: denoted as $\mathcal{G}_s = (V_s, E_s)$, where V_s and E_s are the sets of users and acquaintance relations, respectively. A user $u \in V_s$ can be either a driver or a rider. An edge $u_{ij} \in E_s$ indicates that two users u_i and u_j ($u_i, u_j \in V_s$) are acquainted. Each user u is associated with a set of keywords $\mathcal{K}_u$ that describes the user's personalized interests and features, such as educations, preferences, favorite sports, and so on.
- Drivers: a set of users who offer ridesharing service, denoted as $\mathcal{D}$. Here, $\mathcal{D} \subseteq V_s$. Each driver $d \in \mathcal{D}$ can

be represented by a tuple $d = (l_d^p, l_d^d, t_d^p, \gamma_d, \chi_d, \mathcal{K}_d, \mathcal{V}_d)$, which depicts the current status of driver d for further ridesharing considerations. Here, $l_d^p \in V_n$ and $l_d^d \in V_n$ denote the source and the destination of this driving route, respectively. The tuple d also indicates $t_d^p \in \mathbb{R}^{\geq 0}$ the starting time, $\gamma_d \in \mathbb{R}^{\geq 0}$ the tolerable detour, $\chi_d \in \mathbb{Z}^{\geq 0}$ the number of unoccupied seats, $\mathcal{K}_d$ the driver's interests, and $\mathcal{V}_d \subseteq V_s$ the set of onboard riders;

- Riders: a set of users who request ridesharing service from drivers, denoted as $\mathcal{R}$ ($\mathcal{R} \subseteq V_s$). Each rider $r \in \mathcal{R}$ can be represented by a tuple $r = (l_r^p, l_r^d, t_r^p, t_r^w, \gamma_r, \mathcal{K}_r)$, which describes the requested information of this rider to identify high-quality ridesharing matching. Here, the rider r is located at the pick-up point $l_r^p \in V_n$ at time t_r^p, and desires to drop off at the point $l_r^d \in V_n$. The maximum tolerated waiting time and detour of rider r are $t_r^w \in \mathbb{R}^{\geq 0}$ and $\gamma_r \in \mathbb{R}^{\geq 0}$, respectively. The personalized interests of rider r are $\mathcal{K}_r$.

In practice, since road segments in the city are usually short and road intersections are the most recognizable pick-up and drop-off points for drivers and riders, for simplicity, we follow the existing works [24], [19], [11] assuming that the pick-up and drop-off points are located at the road intersections. Then, we introduce a definition of route schedule, which serves as a basic component of the ridesharing service.

Definition 1 (Route schedule)**.** *Given a driver d with an onboard rider set $\mathcal{V}_d = \{r_1, r_2, \cdots, r_m\}$ ($m \in \mathbb{Z}^+$), the route schedule of d is a time-aware point sequence $\mathcal{S}_d = \langle l_1, l_2, \cdots, l_{2m} \rangle$ with minimum travel cost of $\sum_{i=1}^{2m-1} \pi(l_i, l_{i+1})$, where the road intersection l_1 is the d's source (i.e., $l_1 = l_d^p$), the road intersection l_{2m} is the d's destination (i.e., $l_{2m} = l_d^d$), l_i ($1 < i < 2m$) is the pick-up or drop-off point of r_j where $1 \leq j \leq m$, and $\pi(l_i, l_{i+1})$ is the travel cost between two points l_i and l_{i+1}.*

Note that the route schedule $\mathcal{S}_d$ is valid *if and only if* it follows two conditions: (i) for any rider $r \in \mathcal{V}_d$, the pick-up point l_r^p is ahead of the drop-off point l_r^d in $\mathcal{S}_d$, and (ii) d can serve all riders in $\mathcal{V}_d$ (i.e., d can pick up/drop off the riders in $\mathcal{V}_d$ without violating their waiting time and detour constraints). Additionally, we use $\pi(\cdot, \cdot)$ to represent the travel cost between two points on the road network, and $\pi_v(\cdot, \cdot)$ to represent the actual travel cost for a driver or rider to move from one point to another on the road network according to the driver's route schedule. In this paper, the travel cost can be expressed as the travel distance or the travel time. Since these two measurements can be inter-converted when the travel speed is known, we do not explicitly distinguish them and use travel cost for consistency.

3.3 Matching Utility

Given a driver d and a group of riders (*a.k.a.*, a rider group) $\mathcal{R}_g$, we say the driver d matching the potential ride group $\mathcal{R}_g$ if the following two constraints hold: (i) $|\mathcal{R}_g| \leq \chi_d$, and (ii) there exists a valid route schedule for d to serve the onboard riders in $\mathcal{V}_d$ and the riders in $\mathcal{R}_g$, namely $\mathcal{V}_d \cup \mathcal{R}_g$, according to Definition 1. Although the above two constraints can reduce lots of disqualified matchings, there still exist multiple different choices for ride matching between drivers and riders. Therefore, to quantify the quality of a matching, we formulate the definition of matching utility based on the two important considerations of *social comfort* and *price revenue*.

Social comfort $\varphi(d, \mathcal{R}_g)$. We first introduce the social comfort. In a ridesharing service, riders and the driver often seek for the social comfort among themselves in terms of common interests and social cohesion. For instance, two riders who are friends and have a common interest (e.g., sport) tend to feel comfortable when they share a ride. Thus, following the existing works [17], [16], we define the social comfort for driver d and ride group $\mathcal{R}_g$ as

$$\varphi(d, \mathcal{R}_g) = \frac{\tau(d, \mathcal{R}_g)}{\xi(d, \mathcal{R}_g)}. \tag{1}$$

Here, $\tau(d, \mathcal{R}_g)$ is to measure the common interests of all potential ridesharing users Φ_d, that is, $\Phi_d = \{d\} \cup \mathcal{R}_g \cup \mathcal{V}_d$, including driver d, the existing riders $\mathcal{V}_d$, and also the candidate ride group $\mathcal{R}_g$. We calculate the similarity by a modified Jaccard similarity function $\tau(d, \mathcal{R}_g) = \frac{|\bigcap_{u \in \Phi_d} \mathcal{K}_u| + 1}{|\bigcup_{u \in \Phi_d} \mathcal{K}_u| + 1}$. The social cohesion $\xi(d, \mathcal{R}_g)$ is the maximum social distance between two users in Φ_d (the maximum hops between users Φ_d in social network), denoted as $\xi(d, \mathcal{R}_g) = \max_{u,v \in \Phi_d} dist_{G_s}(u, v)$, where $dist_{G_s}(u, v)$ is the social distance between u and v in social network G_s.

Price revenue $\mu(d, \mathcal{R}_g)$. We next present the price revenue. In a ridesharing service, the price cost and total revenue are critical important for both riders and drivers, respectively. To give a fair price, we adopt the setting of a price scheme following [24]. We assume that each rider can enjoy a certain discount on his/her original travel cost that is proportional to the detour incurred by the driver serving other riders. The fare of a rider r served by a driver d is defined as $\mathcal{F}(r, d) = \pi_v(l_r^p, l_r^d) \cdot f(\Delta) \cdot \eta$, where η is the benefit of unit travel cost, $\Delta = \frac{\pi_v(l_r^p, l_r^d) - \pi(l_r^p, l_r^d)}{\pi(l_r^p, l_r^d)}$ is the actual detour rate, and f: $\mathbb{R}^{[0,+\infty)} \to \mathbb{R}^{(0,1]}$ is the discount function. Thus, for a rider group of multiple users $\mathcal{R}_g$ and the driver d, the price revenue $\mu(d, \mathcal{R}_g)$ is defined as,

$$\mu(d, \mathcal{R}_g) = \frac{\sum_{r \in \mathcal{R}_g} \mathcal{F}(r, d) - \pi_v(l_d^p, l_d^d)}{\mathcal{M}}. \tag{2}$$

Here, $\sum_{r \in R_g} \mathcal{F}(r, d)$ is the total fare for d serving all riders in $\mathcal{R}_g$, $\mathcal{M}$ is the maximum revenue in the historical data for a driver offering a ride, which is used for normalizing $\mu(d, \mathcal{R}_g)$ into $[0, 1]$ and $\pi_v(l_d^p, l_d^d)$ is the total travel cost for driver d serving all riders in $\mathcal{R}_g$. For simplicity, we set the cost of unit travel cost as 1. Note that the ridesharing service platform receives a cut with a fixed rate on the revenue of each driver in practice. Thus, the platform's revenue is proportional to the total revenue of all drivers. In the sequel, we do not distinguish the platform's revenue from drivers' revenue.

Matching utility $\kappa(d, \mathcal{R}_g)$. Integrating the utilities of social comfort and price revenue, the matching utility between the driver d and the rider group $\mathcal{R}_g$ is defined as

$$\kappa(d, \mathcal{R}_g) = \alpha \times \varphi(d, \mathcal{R}_g) + (1 - \alpha) \times \mu(d, \mathcal{R}_g), \tag{3}$$

where the parameter $\alpha \in [0, 1]$ is used to make a trade-off of user preferences between social comfort $\varphi(d, \mathcal{R}_g)$ and price

revenue $\mu(d, \mathcal{R}_g)$.

3.4 Problem Formulation

Next, we consider a set of drivers $\mathcal{D}$ and a set of riders $\mathcal{R}$, and define a matching plan between $\mathcal{D}$ and $\mathcal{R}$, in the aspect of drivers and riders matching *globally* on a ridesharing service platform.

Definition 2 (Matching plan)**.** *Given a driver set $\mathcal{D}$ and a rider set $\mathcal{R}$, a matching plan is defined as $\mathcal{P} = \{(d, \mathcal{R}_g) | d \in \mathcal{D}, \mathcal{R}_g \subseteq \mathcal{R}, |\mathcal{R}_g| \leq \chi_d\}$ where for any two matchings $(d, \mathcal{R}_g) \in \mathcal{P}$ and $(d', \mathcal{R}'_g) \in \mathcal{P}$ with $d \neq d'$, it holds for $\mathcal{R}_g \cap \mathcal{R}'_g = \emptyset$.*

Note that in a matching plan $\mathcal{P}$ in Def. 2, each rider can be assigned to at most one driver. Meanwhile, each driver can be assigned a rider set $\mathcal{R}_g$ where the size upper bound of $\mathcal{R}_g$ is the number of available seats of d (i.e., $|\mathcal{R}_g| \leq \chi_d$). The matching utility of $\mathcal{P}$ can be calculated as

$$Score(\mathcal{P}) = \sum_{(d, \mathcal{R}_g) \in \mathcal{P}} \kappa(d, \mathcal{R}_g). \tag{4}$$

According to the matching plans and the corresponding matching utilities, we formulate as follows the URM-problem studied in this paper.

Problem 1 (URM-problem)**.** *Given a set of drivers $\mathcal{D}$ and a set of riders $\mathcal{R}$, the* URM-problem *aims to find the best matching plan $\mathcal{P}$ with the largest matching utility $Score(\mathcal{P})$, that is, there exists no other matching plan $\mathcal{P}' \neq \mathcal{P}$ such that $Score(\mathcal{P}') > Score(\mathcal{P})$.*

3.5 Problem Analysis

In the following, we analyze the hardness of the URM-problem in Theorem 1.

Theorem 1. *The* URM-problem *is NP-hard.*

Proof. Please refer to Appendix A.1. □

In this paper, we study how to solve the URM-problem efficiently and effectively. We aim at developing optimal and scalable solutions that have short response times and high-quality answers. This is mainly achieved by several efficient algorithms, which are presented in Sections 4 and 5.

4 Exact Utility-Aware Matching

In this section, we first reformulate the URM-problem in an optimization way, which is addressed by integer linear programming. Then, we present a bipartite matching algorithm to find optimal answers with the largest matching utilities.

4.1 Integer Linear Programming Approach

We first reformulate URM-problem to be solved by integer linear programming. Given a set of drivers $\mathcal{D}$ and a set of riders $\mathcal{R}$, the URM-problem aims to find out an optimal matching plan $\mathcal{P}$ in $\mathcal{D} \times \mathcal{R}_G$ where $\mathcal{R}_G$ (which can be found by DFS) denotes a set of rider groups that can be served by the drivers in $\mathcal{D}$. For a driver $d_i \in \mathcal{D}$ ($i \in [1, |\mathcal{D}|]$) and a rider group $\mathcal{R}_{g_j} \in \mathcal{R}_G$ ($j \in [1, |\mathcal{R}_G|]$), we use the decision variable $x_{ij} \in \{0, 1\}$ to indicate the matching of d_i and $\mathcal{R}_{g_j}$ where $x_{ij} = 1$ denotes the matching of d_i and $\mathcal{R}_{g_j}$ is valid and $x_{ij} = 0$ otherwise. We use the decision variable $y_{ik} \in \{0, 1\}$ ($i \in [1, |\mathcal{D}|]$ and $k \in [1, |\mathcal{R}|]$) to reveal the matching of d_i and r_k where $y_{ik} = 1$ denotes that d_i serves r_k and $y_{ik} = 0$ otherwise. Hence, the objective function for solving the URM-problem can be formulated as follows:

$$\textbf{max} \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{R}_G|} w_{ij} \cdot x_{ij}$$

$s.t.$,

$$x_{ij} \in \{0, 1\}, \tag{5}$$

$$y_{ik} \in \{0, 1\}, \tag{6}$$

$$w_{ij} = \kappa(d_i, \mathcal{R}_{g_j}), \tag{7}$$

$$\sum_{i=1}^{|\mathcal{D}|} y_{ik} \leq 1, \forall k \in \{1, 2, \cdots, |\mathcal{R}|\}, \tag{8}$$

$$\sum_{j=1}^{|\mathcal{R}_G|} x_{ij} \leq 1, \forall i \in \{1, 2, \cdots, |\mathcal{D}|\}. \tag{9}$$

Here, constraint (5) and constraint (6) simply enforce the binary nature of x_{ij} and y_{ik}; constraint (7) calculates the matching utility of d_i and $\mathcal{R}_{g_j}$; constraint (8) implies that each rider can match at most with one driver; and constraint (9) requires that each driver can serve at most one rider group. The integer linear programming is the basic approach for solving the URM-problem and is effective only when the number of drivers and riders is in a small size. As proved in Theorem 1, however, the URM-problem is NP-hard. If there are many drivers and riders, solving the URM-problem by integer linear programming is computationally intensive and thus it cannot work well on real applications with more than thousands of drivers and riders. Hence, we should propose solutions that are more efficient to solve this problem.

Algorithm 1: Basic Bipartite Matching

Data: a set of drivers $\mathcal{D}$, a set of riders $\mathcal{R}$
Result: the best matching plan $\mathcal{P}$
1 Initialize a driver-rider set $\mathcal{DR}$;
`/* Filtering (lines 2--4): filter out invalid driver-rider matching. */`
2 **for** *each pair* $(d, r) \in \mathcal{D} \times \mathcal{R}$ **do**
3 **if** *the matching of d and r is valid* **then**
4 Add the pair (d, r) into $\mathcal{DR}$;
5 Build the weighted bipartite graph $\mathcal{G}_b$ based on $\mathcal{DR}$;
`/* Verification (line 6): invoke the KM algorithm to find the best matching plan. */`
6 Find the best matching plan $\mathcal{P}$ in $\mathcal{G}_b$ using KM algorithm [29], [20];
7 **return** $\mathcal{P}$;

4.2 Bipartite Matching Approach

In this section, we solve the URM-problem by bipartite matching, which finds the best matching plan between drivers and riders. We model the underlying setting of the

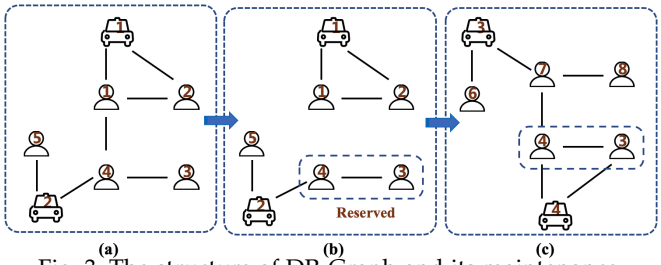


Fig. 3: The structure of DR-Graph and its maintenance.

URM-problem as a weighted bipartite graph $\mathcal{G}_b = (\mathcal{D}, \mathcal{R}, \mathcal{E})$ that contains a set of drivers $\mathcal{D}$, a set of riders $\mathcal{R}$, and a set of edges $\mathcal{E}$. Each edge $e \in \mathcal{E}$ linking a driver $d \in \mathcal{D}$ and a rider $r \in \mathcal{R}$ exists if the matching of d and r is valid and associated with a weight indicating the utility for d serving r. Hence, solving the URM-problem means finding the best matching plan $\mathcal{P}$ over $\mathcal{G}_b$ such that the overall utility of $\mathcal{P}$ is maximized.

Basic Bipartite Ride Matching. We illustrate a basic algorithm of bipartite matching on $\mathcal{G}_b$, which serves as a general framework for finding the best driver-rider matching plan as shown in Algorithm 1. The framework is composed of two steps including *filtering* and *verification*. In short, we first find all valid driver-rider matchings $\mathcal{DR}$ by checking whether the drivers can serve the riders (lines 2–4). After that, we build the weighted bipartite group $\mathcal{G}_b$ based on $\mathcal{DR}$ (line 5). Finally, we invoke the KM algorithm [20], [29] to find the best matching plan $\mathcal{P}$ (line 6). Algorithm 1 can correctly find the optimal matching plan when each driver can serve at most one rider. However, it cannot tackle the case when a driver serves multiple riders optimally.

Next, we present a novel index structure, called Driver-Rider Graph (DR-Graph), to address this issue. We give the definition of the DR-graph as follows.

Definition 3 (DR-Graph)**.** *A DR-Graph, denoted as $\mathcal{G}_h = (V_h, E_h)$, is a graph where each vertex $v \in V_h$ denotes a driver or a rider, and an edge $e = (v, u) \in E_h$ linking two vertices v and u exists if the matching of v and u is valid.*

Figure 3(a) illustrates the graph structure of DR-Graph. Note that the matching of v and u in Definition 3 is twofold: (1) if v and u are riders, the matching (v, u) indicates that at least one nearby driver can serve both v and u; (2) if v is a driver and u is a rider, the matching (v, u) indicates that v can serve u.

DR-Graph Construction and Maintenance. In the following, we present how to construct the DR-Graph and maintain it with dynamic updates. Given a set of drivers $\mathcal{D}$ and a set of riders $\mathcal{R}$, an essential task for building a DR-Graph is to find both kinds of edges, that is, driver-rider and rider-rider matchings. For the driver-rider matching, it is easy to identify if the driver can serve the rider without violating their trip constraints (e.g., tolerable waiting time and detour ratio). For the rider-rider matching, given two riders r_1 and r_2, we assume that there exists an ideal driver d whose source and destination are the same as those of riders r_1 or r_2. Then, the rider-rider matching (r_1, r_2) exists if an ideal driver d can serve riders r_1 and r_2 without violating the constraints of r_1 and r_2. In other words, at least one of the following four schedules for d serving r_1 and r_2 is valid,

$$\begin{aligned}\mathcal{S}_d = \{&< l^p_{r_1}, l^p_{r_2}, l^d_{r_2}, l^d_{r_1} >,\\ &< l^p_{r_1}, l^p_{r_2}, l^d_{r_1}, l^d_{r_2} >,\\ &< l^p_{r_2}, l^p_{r_1}, l^d_{r_1}, l^d_{r_2} >,\\ &< l^p_{r_2}, l^p_{r_1}, l^d_{r_2}, l^d_{r_1} >\}.\end{aligned}$$

On the other hand, the DR-Graph can be also adapted to the dynamic updating setting of our URM-problem, which maintains the matching computations performed up-to-now and uses them efficiently for newly joined and unmatching drivers/riders in the next batch. Specifically, we dynamically insert and remove edges for the newly arrived drivers and riders.

Example 1. *We illustrate an example of DR-Graph and its dynamic maintenance in Figure 3. Figure 3(a) shows an initial DR-Graph, which consists of five riders $r_1 \sim r_5$ and two drivers d_1 and d_2. The edges have two kinds of driver-rider and rider-rider matchings. Figure 3(b) depicts the best matching plan of the DR-Graph in Figure 3(a). The driver d_1 matches the rider group $\{r_1, r_2\}$ and the driver d_2 matches the rider r_5. Note that the rider-rider matching between r_3 and r_4 remains for the next round matching. Assume that three new riders r_6, r_7, r_8 and two new drivers d_3, d_4 arrive in the DR-Graph. We accordingly update the DR-Graph as shown in Figure 3(c). As the matching (r_3, r_4) is still valid for matching, we do not need to build this matching from scratch and thus save computation costs via this method of DR-Graph maintenance.*

Although the DR-Graph can keep all matching information among drivers and riders, it is not precise enough to infer the matching between a driver and a group of riders. Thus, we propose an advanced graph model DRg-Graph based on the DR-Graph.

DRg-Graph Construction. The DRg-Graph is formulated by the three entities of drivers, riders, and rider groups. We can construct the DRg-Graph based on the DR-Graph as follows. We adopt the depth-first search algorithm to quickly find a set of rider groups $\mathcal{R}^d_g$ that can be served by a driver d on the DR-Graph. We treat each set of rider groups as a super vertex in DRg-Graph. Making use of these rider groups, we then define the DRg-Graph as $\mathcal{G}_B = (\mathcal{D}, \mathcal{R}_G, \mathcal{E}^+)$ to support finding the best matching plan that allows a driver to serve multiple riders, in which $\mathcal{R}_G = \bigcup_{d \in \mathcal{D}} \mathcal{R}^d_g$, $\mathcal{E}^+$ contains all edges linking $d \in \mathcal{D}$ and $\mathcal{R}_g \in \mathcal{R}_G$. For each bipartite graph $\mathcal{G}_b$ enumerated from $\mathcal{G}_B$, we invoke the KM algorithm to find the best matching plan $\mathcal{P}_b$. Then the best matching plan with the maximum utility is returned as the final result. However, this approach needs to compute the matching utilities for all ridesharing match plans. To further improve the processing performance, we present a utility upper bound in Lemma 1 to stop the computation process early, guaranteed by Theorem 2. Here, Lemma 1 can be easily proved and thus omitted.

Lemma 1. *Given a bipartite graph $\mathcal{G}_b$, the utility upper bound of a matching plan $\mathcal{P}_b$ derived from $\mathcal{G}_b$ is*

$$Score_{ub}(\mathcal{P}_b) = \sum_{d \in \mathcal{D}} \mathbf{max}_{\mathcal{R}_g \in \mathcal{R}^d_g} \kappa(d, \mathcal{R}_g), \qquad (10)$$

Algorithm 2: Bipartite Matching Algorithm (**BMA**)

Data: a set of drivers $\mathcal{D}$, a set of riders $\mathcal{R}$, and a DR-Graph $\mathcal{G}_h$
Result: the final matching plan $\mathcal{P}$

1 $queue \leftarrow$ NewPriorityQueue();
2 **for** *each driver* $d \in \mathcal{D}$ **do**
3 Find the rider group set $\mathcal{R}_d^+$ in $\mathcal{G}_h$ using DFS;
4 $\mathcal{B}_b \leftarrow \mathcal{B}_b \cup \mathcal{R}_d^+$;
5 Build the super bipartite graph $\mathcal{G}_B$;
6 **for** *each bipartite graph* $\mathcal{G}_b$ *in* $\mathcal{G}_B$ **do**
7 Compute the utility upper bound $Score_{ub}(\mathcal{G}_b)$;
8 $queue$.Enqueue($\mathcal{G}_b$, $Score_{ub}(\mathcal{G}_b)$);
9 **while** $queue \neq \emptyset$ **do**
10 $\mathcal{G}_b' \leftarrow queue$.Dequeue();
11 Find the best matching plan $\mathcal{P}'$ of $\mathcal{G}_b'$ using KM [29];
12 **if** $Score(\mathcal{P}') \geq Score(\mathcal{P})$ **then**
13 $\mathcal{P} \leftarrow \mathcal{P}'$;
14 $\mathcal{G}_b'' \leftarrow queue$.First();
15 **if** $Score(\mathcal{P}) \geq Score_{ub}(\mathcal{G}_b'')$ **then**
16 **break**;
17 **return** $\mathcal{P}$;

where $\mathcal{R}_g^d$ is a set of rider groups that can be served by driver d.

Theorem 2. *Given a queue of bipartite graphs $\mathcal{B}$ = ($\mathcal{G}_{b_1}$, $\mathcal{G}_{b_2}$, $\cdots$, $\mathcal{G}_{b_m}$), where the bipartite graph $\mathcal{G}_{b_i}$ is sorted in descending order of the utility upper bound $Score_{ub}(\mathcal{P}_{b_i})$. If $Score(\mathcal{P}_{b_i}) \geq Score_{ub}(\mathcal{P}_{b_{i+1}})$, then $\mathcal{P}_{b_i}$ is the optimal matching plan from $\mathcal{B}$.*

Proof. Please refer to Appendix A.2. □

BMA Algorithm. Leveraging the DR-graph and DRg-Graph, we present the bipartite matching algorithm (BMA) for finding the best matching plan. Algorithm 2 describes the details of the bipartite matching algorithm. We first initializes a priority queue $queue$ where each entry is a bipartite graph sorted in the decreasing order of utility upper bounds (line 1). Then, the rider groups that can be served by each driver in $\mathcal{D}$ are calculated using the DFS algorithm and stored in $\mathcal{B}_b$ (lines 2-4). With these rider groups, we construct a DRg-Graph with rider groups $\mathcal{G}_B$ (line 5) and computes the utility upper bound $Score_{ub}(\mathcal{G}_b)$ for each bipartite graph $\mathcal{G}_b$ in $\mathcal{G}_B$. The algorithm adds each $\mathcal{G}_b$ into $queue$ according to its $Score_{ub}(\mathcal{G}_b)$. Afterwards, it dequeues the bipartite graph $\mathcal{G}_b'$ from $queue$ and computes the best matching plan $\mathcal{P}'$ of $\mathcal{G}_b'$ by KM algorithm [29], [20]. Here, we use the matching plan $\mathcal{P}$ to store the current best matching plan (lines 12-13). If the utility of $\mathcal{P}$ is larger than the utility upper bound of the next processing bipartite graph in $queue$, the matching plan $\mathcal{P}$ is returned as the final answer (lines 14-17).

Example 2. *We illustrate a running example of applying the bipartite matching approach in Algorithm 2 on DR-graph $\mathcal{G}_h$ in Figure 4. The drivers and riders are shown in $\mathcal{G}_h$ in Figure 4(a). Given this DR-Graph $\mathcal{G}_h$, we invoke the DFS algorithm to construct* DRg-Graph *$\mathcal{G}_B$ as shown in Figure 4(b). Based on the derived* DRg-Graph *$\mathcal{G}_B$, we enumerate all possible cases of bipartite graphs and sort them in the descending order of their utility upper bounds. We sequentially calculate the utility of the best matching plan for each bipartite graph. For instance, when the best utility of the bipartite graph in Case 1 is calculated, the best matching utility is shown as 10+9+5=24. It is higher than the utility upper bound of that in Case 2 (i.e., 22), which leads to a stop of the calculation by Theorem 2. Therefore, the best matching plan in Case 1 is returned as the final matching answer.*

Algorithm 3: Fast Matching Algorithm (**FMA**)

Data: a set of drivers $\mathcal{D}$, a set of riders $\mathcal{R}$, and a DRg-Graph $\mathcal{G}_B$
Result: the final matching plan $\mathcal{P}$

1 Initialize the best matching plan $\mathcal{P}$ from $\mathcal{G}_B$ by KM;
2 **for** *each rider group* $\mathcal{R}_g$ *in* $\mathcal{G}_B$ **do**
3 Find the drivers $\mathcal{D}_{\mathcal{R}_g}$ in $\mathcal{P}$ that can serve $\mathcal{R}_g$;
4 **if** *no subset of* $\mathcal{R}_g$ *appears in* $\mathcal{P}$ **then**
5 **for** *each driver* d *in* $\mathcal{D}_{\mathcal{R}_g}$ **do**
6 $\mathcal{R}_g \leftarrow$ the rider group served by d;
7 $\mathcal{D}' \leftarrow$ the drivers in $\mathcal{D}$ but not in $\mathcal{P}$;
8 $d' \leftarrow \mathbf{argmax}_{d' \in \mathcal{D}'} \kappa(d', \mathcal{R}_g)$;
9 $\delta' \leftarrow \kappa(d', \mathcal{R}_g) - \kappa(d, \mathcal{R}_g)$;
10 $\delta \leftarrow \mathbf{max}\{\delta', \delta\}$;
11 **else**
12 $\mathcal{D}_p \leftarrow$ the drivers in $\mathcal{P}$ serving subset of $\mathcal{R}_g$;
13 $\mathcal{H} \leftarrow$ the rider groups in $\mathcal{G}_B$ but not in $\mathcal{P}$;
14 **if** $\mathcal{D}_{\mathcal{R}_g}$ *is not empty* **then**
15 $d' \leftarrow \mathbf{argmax}_{d \in \mathcal{D}_{\mathcal{R}_g}} \kappa(d, \mathcal{R}_g)$;
16 $\delta' \leftarrow \mathbf{max} \sum_{d \in \mathcal{D}_p \setminus \{d'\}, \mathcal{R}_g \in \mathcal{H}} \kappa(d, \mathcal{R}_g)$;
17 $\delta \leftarrow \kappa(d', \mathcal{R}_g) + \delta' - \sum_{d \in \mathcal{D}_p} \kappa(d, \mathcal{R}_g)$;
18 **if** $\delta > 0$ **then**
19 Update $\mathcal{P}$ with $\mathcal{R}_g$ accordingly;
20 **return** $\mathcal{P}$;

Complexity analysis. The time cost of BMA algorithm consists of two parts: (i) *DRg-Graph construction*, the time complexity for constructing DRg-Graph is $O(n(m+n)+n\beta|R_g^d|)$ (denoted as DRG) where m is the number of riders, n is the number of drivers, and $|R_g^d|$ is the number of rider groups in R_g^d; and (ii) *matching plan finding*, the time complexity for finding the optimal matching plan is $\beta\overline{m}^3$ where β is the number of bipartite graphs enumerated from $\mathcal{G}_B$, $\overline{m}$ is the maximum number of drivers or ride groups in $\mathcal{G}_b'$, and $O(\overline{m}^3)$ is the time complexity of KM algorithm. Therefore, the time complexity of the BMA is $O(DRG + \beta\overline{m}^3)$.

5 Efficient Utility-Aware Matching

In this section, we develop two efficient solutions for the URM-problem. We propose a greedy algorithm for quickly finding ridesharing matching answers. Moreover, to balance the efficiency and effectiveness, we propose an ϵ-refining algorithm with quality guaranteed.

5.1 Fast Matching Approach

Although the bipartite matching algorithm finds the best matching plan correctly, it is very time consuming to enumerate all the possible cases in the worst. To improve the efficiency, we propose an efficient fast matching approach (FMA) in Algorithm 3 to solve the URM-problem.

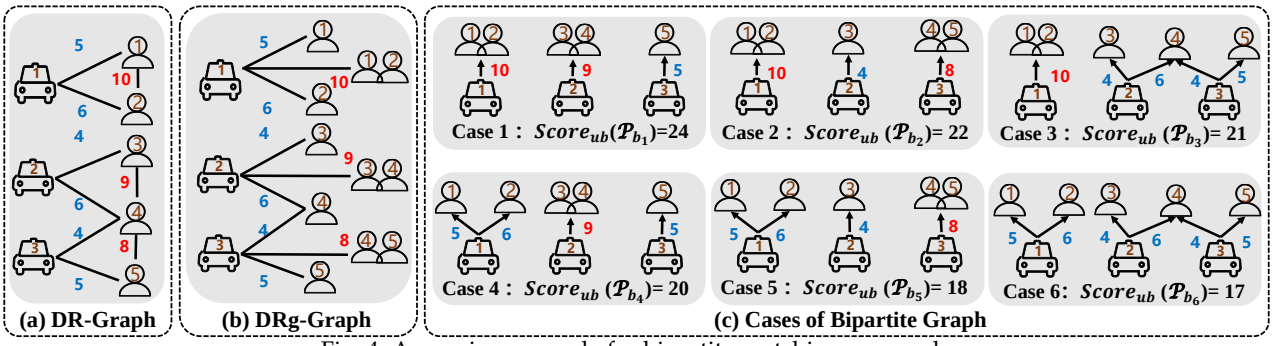


Fig. 4: A running example for bipartite matching approach.

Overview. The general idea of our FMA approach in Algorithm 3, is that given a set of drivers $\mathcal{D}$, a set of riders $\mathcal{R}$, and a DRg-Graph $\mathcal{G}_B$, we first use the KM algorithm to initialize the best matching plan $\mathcal{P}$ over $\mathcal{G}_B$, where each driver serves at most one rider. Then, we invoke the DFS algorithm to find all rider groups over $\mathcal{G}_h$. For each rider group $\mathcal{R}_g$, we continuously update the rider in $\mathcal{P}$ with $\mathcal{R}_g$ to improve the overall utility. Finally, the updated matching plan $\mathcal{P}$ is returned as the final answer. In general, the updates of the rider group of d in $\mathcal{P}$ given $\mathcal{R}_g$ are twofold:

- **CASE 1**. If no subset of $\mathcal{R}_g$ appears in $\mathcal{P}$, we attempt to find a driver d in $\mathcal{P}$ to serve $\mathcal{R}_g$ and find a driver in $\mathcal{D}$ (but not in $\mathcal{P}$) to serve d's matched rider group $\mathcal{R}_d$ in $\mathcal{P}$ such that the utility increment for the rider group exchange is maximized.
- **CASE 2**. If there exist some subsets of $\mathcal{R}_g$ appearing in $\mathcal{P}$, we attempt to find a driver d in $\mathcal{P}$ to serve $\mathcal{R}_g$ instead of the drivers $\mathcal{D}'$ serving the subsets of $\mathcal{R}'$ in $\mathcal{P}$. Meanwhile, we match each driver $d \in \mathcal{D}'$ with a suitable rider group of $\mathcal{G}_B$ that is not in $\mathcal{P}$, such that the utility increment for the rider group exchange is maximized.

FMA Algorithm. Algorithm 3 describes the procedures for the fast matching algorithm. The algorithm takes the input of a set of drivers $\mathcal{D}$, a set of riders $\mathcal{R}$, and a DRg-Graph $\mathcal{G}_B$. It first invokes the KM algorithm to find the initial matching result where each driver serves one rider at most (line 1). Then, for each rider group $\mathcal{R}_g$ in $\mathcal{G}_B$, it finds the drivers $\mathcal{D}_{\mathcal{R}_g}$ in $\mathcal{P}$ that can serve $\mathcal{R}_g$ (line 3). The algorithm next identifies which exchange the update falls into according to whether the subset of $\mathcal{R}_g$ appears in $\mathcal{P}$. If there is no subset of $\mathcal{R}_g$ appearing in $\mathcal{P}$, the algorithm adopts the exchange of CASE 1 (lines 4-10); otherwise, it finds the driver set $\mathcal{D}_{\mathcal{P}}$ that can serve the subset of $\mathcal{R}_g$ and the rider groups $\mathcal{H}$ in $\mathcal{G}_B$ but not in $\mathcal{P}$. After that, it selects the driver $d' \in \mathcal{D}_{\mathcal{P}}$ who matches $\mathcal{R}_g$ with the maximum utility. Since the riders in $\mathcal{R}_g$ are served by d', it needs to match each driver in $\mathcal{D}_{\mathcal{P}} \setminus \{d'\}$ with a suitable rider group in $\mathcal{H}$ such that the overall utility of $\mathcal{D}_{\mathcal{P}} \setminus \{d'\}$ is maximized (lines 12-17). Finally, if the utility increment δ is larger than zero, it executes the rider group exchange in $\mathcal{P}$ accordingly (lines 18-19). The process of rider group exchange executes until all rider groups in $\mathcal{G}_B$ are attempted. The final matching plan $\mathcal{P}$ is returned as the answer (line 20).

Example 3. *We illustrate a running example to introduce the general ideas of our fast matching approach in Figure 5. Figure 5(a) shows an input* DRg-Graph *of drivers, riders, and rider groups. Figure 5(b) depicts an initial matching plan in which each driver serves at most one rider. Figures 5(c)∼5(e) illustrate the process of matching plan updates. In Figures 5(c)∼5(e), we use the rider groups* $\{r_1, r_2\}$, $\{r_3, r_4\}$, *and* $\{r_6, r_7\}$ *to update the initial matching plan, respectively. Specifically, the utility increment of the updated matching plan in Figure 5(c) is* $8-6=2$ *by changing* r_1 *with* $\{r_1, r_2\}$. *Similarly, the utility increment in Figure 5(d) is* $11-8=3$ *by changing* r_4 *with* $\{r_3, r_4\}$, *the utility increment in Figure 5(e) is* $11-8+6-7=2$ *by changing* r_7 *with* $\{r_6, r_7\}$.

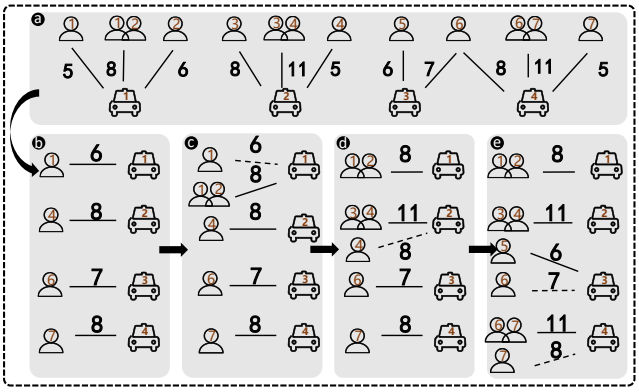


Fig. 5: A running example for fast matching approach.

Complexity analysis. The time complexity of the fast matching approach is $O(m^3 + \nu^2 k)$, where m is the number of riders, ν is the number of rider groups, and k is the number of drivers that matches the rider groups. Here, $O(m^3)$ is the complexity of the KM algorithm.

5.2 ϵ-Refining Matching Approach

Although the fast matching algorithm can greatly improve processing efficiency, it cannot guarantee the quality of the matching result. To address this issue, we propose an efficient ϵ-refining algorithm for balancing the answer quality and processing efficiency, ensuring the matching answer $\mathcal{P}$ achieves an ϵ-approximation to the matching utility of optimal answer $\mathcal{P}^*$. Here, the value of ϵ is a user-defined parameter of practicality.

5.2.1 Bounded DRg-Graph and Pruning Bounds

Bounded DRg-Graph. We begin with a new definition of the bounded DRg-Graph. The index structure of the bounded DRg-Graph is built upon the DRg-Graph in Definition 4. Thus, the bounded DRg-Graph also consists of drivers and rider groups but keeps records of the upper bound and lower bound of utilities (instead of the exact utilities), which

bounds are used for pruning disqualified matchings and reducing computation costs.

Definition 4 (Bounded DRg-Graph). *A bounded* DRg-Graph $\mathcal{BG} = (\mathcal{D}, \mathcal{R}_{\mathcal{D}}^{+}, \mathcal{E}^{+})$ *is a* DRg-Graph. *Each edge* $e = (d, \mathcal{R}_g) \in \mathcal{E}^{+}$ *linking a driver* d *and a rider group* $\mathcal{R}_g$ *is with a utility lower bound* $\kappa_l(d, \mathcal{R}_g)$ *and a utility upper bound* $\kappa_u(d, \mathcal{R}_g)$.

Next, we introduce two useful properties in Theorems 3 and 4 to calculate the lower bound $\kappa_l(d, \mathcal{R}_g)$ and the upper bound $\kappa_u(d, \mathcal{R}_g)$ for each matching between drivers and rider groups, that is, $\kappa_l(d, \mathcal{R}_g) \leq \kappa(d, \mathcal{R}_g) \leq \kappa_u(d, \mathcal{R}_g)$. In the following, we still consistently denote the all potential ridesharing users for driver d as $\Phi_d = \{d\} \cup \mathcal{R}_g \cup \mathcal{V}_d$.

Theorem 3. *Given a matching of a driver* d *and a rider group* $\mathcal{R}_g$*, the utility lower bound of the matching* $(d, \mathcal{R}_g)$ *is*

$$\kappa_l(d, \mathcal{R}_g) = \alpha \cdot \frac{|\bigcap_{u \in \Phi_d} \mathcal{K}_u| + 1}{\mathcal{SD}_{max} \cdot \sum_{u \in \Phi_d} |\mathcal{K}_u| + 1} + (1 - \alpha) \cdot \frac{\sum_{r \in \mathcal{R}_g} \pi_v(l_r^p, l_r^d) \cdot f(\Delta_{max}) \cdot \eta - \pi_v(l_d^p, l_d^d)}{\mathcal{M}}, \tag{11}$$

where $\mathcal{SD}_{max}$ *is the maximum social distance between users.*

Proof. Please refer to Appendix A.3 □

Theorem 4. *Given a matching of a driver* d *and a rider group* $\mathcal{R}_g$*, the utility upper bound of the matching* $(d, \mathcal{R}_g)$ *is*

$$\kappa_u(d, \mathcal{R}_g) = \alpha \cdot \frac{\mathbf{max}_{u \in \Phi_d} |\mathcal{K}_u| + 1}{|\bigcup_{u \in \Phi_d} \mathcal{K}_u| + 1} + (1 - \alpha) \cdot \frac{\sum_{r \in \mathcal{R}_g} \mathcal{F}(r, d) - \pi_v(l_d^p, l_d^d)}{\mathcal{M}}. \tag{12}$$

Proof. Please refer to Appendix A.4 □

In Theorems 3 and 4, it is time consuming to calculate the travel cost $\pi(l_r^p, l_r^d)$ frequently. In practice, for bound computations, we can use the travel cost lower bound $\pi_l(l_r^p, l_r^d)$ and upper bound $\pi_u(l_r^p, l_r^d)$ given in Lemma 2 instead of the travel cost $\pi(l_r^p, l_r^d)$. In the following, we propose a novel grid-based index to accelerate the computations of upper and lower bounds.

Grid-based Distance Index. We design a novel grid-based distance index to estimate the travel cost upper and lower bounds. Figure 6(a) illustrates the data structure of our grid-based distance index. We partition the road network into a set of continuous cells, according to the number of road intersections within a cell, where each cell is represented by an internal entry. Each cell contains a fixed number of road intersections at most. Besides the grid-partitioned map in Figure 6(a), it is also equipped with two distance matrices of grids M_g and cells M_c, as shown in Figures 6(b) and 6(c), respectively. The grid-based matrix M_g is an internal distance matrix, which keeps the distance between any two cells' center road intersections. The cell-based matrix M_c is an external distance matrix, which keeps the distance between the center road intersection and the road intersection within a cell. Note that the grid-based index and two distance matrices M_g, M_c only need to be calculated once offline. Making use of M_g and M_c, we can quickly calculate the travel cost upper bound $\pi_u(l_r^p, l_r^d)$ and the travel cost lower bound $\pi_l(l_r^p, l_r^d)$, guaranteed by the following Lemma 2.

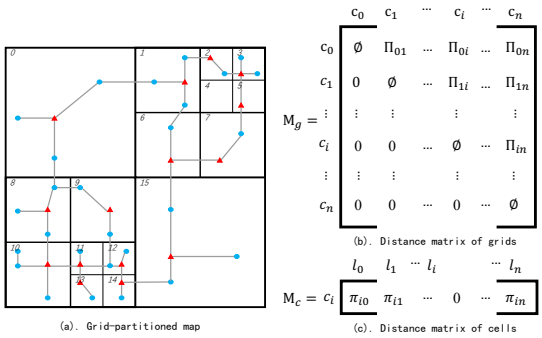


Fig. 6: The structure of the grid-based index. Figure 6(a) is the partition of the road network, the centre points in each cell are marked in red and the other points are marked in blue. Figures 6(b) and 6(c) are two distance matrices.

Lemma 2. *Given two road intersections* l_i *and* l_j*, assume that* l_i *locates in cell* c_k *and* l_j *locates in cell* c_m*. The travel cost upper bound* $\pi_u(l_i, l_j)$ *and the travel cost lower bound* $\pi_l(l_i, l_j)$ *are* $\pi_u(l_i, l_j) = \Pi_{km} + \pi_{ki} + \pi_{mj}$ *and* $\pi_l(l_i, l_j) = max(0, \Pi_{km} - \pi_{ki} - \pi_{mj})$*, respectively.* Π_{km} *is the distance between cells* c_k *and cell* c_m.

Proof. The proof can be easily completed through the triangle inequality. Thus, we omit the proof here. □

5.2.2 ϵ-Refining Algorithm

In the following, we state how to use bounded DRg-Graph to find out an ϵ-approximation matching answer. In general, adopting the bounded DRg-Graph, we can quickly compute the utility upper and lower bound for an optimal matching plan P^* based on Theorem 5. We iteratively calculate the real utility of each matching and use it to update the utility upper and lower bounds for the matching plan until the ratio between the utility upper bound and the utility lower bound is less than the threshold ϵ. Finally, the matching plan with the lower bound is returned as the final answer.

Bounds for $Score(\mathcal{G}_B)$. We first introduce the upper and lower bounds for $Score(\mathcal{G}_B)$ as follows.

Theorem 5. *The utility upper bound* $Score_{ub}(\mathcal{G}_B)$ *and utility lower bound* $Score_{lb}(\mathcal{G}_B)$ *for the matching plan derived from the bounded* DRg-Graph *are*

$$Score_{ub}(\mathcal{G}_B) = \sum_{d \in \mathcal{D}} \mathbf{max}_{\mathcal{R}_g \in \mathcal{R}_g^d} \{\kappa_u(d, \mathcal{R}_g)\} \tag{13}$$

and

$$Score_{lb}(\mathcal{G}_B) = \sum_{i=1}^{|\mathcal{D}|} \mathbf{max}_{d_i \in \mathcal{D}} \{\kappa_l(d_i, \mathcal{R}_g^i)\} \tag{14}$$

where $\mathcal{R}_g^i \in \mathcal{R}_g^{d_i}$, $\mathcal{R}_g^j \in \mathcal{R}_g^{d_j}$ *and* $\cup_{j=1}^{i} \{\mathcal{R}_g^i \cap \mathcal{R}_g^j\} = \emptyset$.

Proof. Please refer to Appendix A.5 □

Overview. Based on the bounded DRg-Graph, the general process of ϵ-refining algorithm is that we use the real utility of each matching to iteratively update the matching utility

Algorithm 4: ϵ-Refining Algorithm (**eRA**)

```
Data: a DRg-Graph G_B, a threshold ε
Result: the final matching plan P
1  queue ← NewPriorityQueue();
2  for each edge e in G_B do
3  |  queue.Enqueue(e, κ_u(d, R_g) − κ_l(d, R_g));
4  ratio ← Score_ub(G_b)/Score_lb(G_b);
5  while ratio > ε and queue ≠ ∅ do
6  |  e ← queue.Dequeue();
7  |  Update G_B with κ(d, R_g) of e;
8  |  ratio ← Score_ub(G_B)/Score_lb(G_B);
9  if ratio ≤ ε then
10 |  Compute an ε-approximation plan P with
   |   Score_lb(G_B);
11 else
12 |  Compute P over G_B by BMA in Algorithm 2;
13 return P;
```

upper and lower bounds. In each iteration of updating, we compare the ratio of $Score_{ub}(\mathcal{G}_B)/Score_{lb}(\mathcal{G}_B)$ with the threshold ϵ. If the ratio is less than ϵ, we return the matching plan with the utility lower bound $Score_{lb}(\mathcal{G}_B)$ as the final answer. Note that we calculate the real utility of matching in descending order of the difference between $Score_{ub}(\mathcal{G}_B)$ and $Score_{lb}(\mathcal{G}_B)$, because it can quickly narrow down the gap between $Score_{ub}(\mathcal{G}_B)$ and $Score_{lb}(\mathcal{G}_B)$ and thus accelerate the termination of the search process.

eRA Algorithm. Algorithm 4 introduces the pseudo code of ϵ-refining algorithm. It first builds the priority queue to store the matchings in the decreasing order by the difference between the utility upper bound and the utility lower bound (line 1). For each matching in the bounded DRg-Graph, if the matching linked driver can serve the riders in the linked rider group, we put the matching into the priority queue according to the utility bound difference (lines 2-3). As the upper and lower bounds of all matchings are known in advance, the ratio between the utility upper bound and the utility lower bound of the best matching plan derived from bounded DRg-Graph can be calculated. If it does not exceed the threshold ϵ, the matching plan corresponding to the utility lower bound will be returned as the final result (lines 5-10). Otherwise, we dequeue and calculate the real utility of the matching on the top of priority queue, then update the utility upper and lower bounds using the real matching utility (line 7). If all the matchings in the queue cannot satisfy the early stop condition, we use the BMA in Algorithm 2 to calculate the optimal matching plan (line 12).

Example 4. *Figure 7 shows an example of ϵ-refining approach where ϵ = 1.2. In Figure 7(a), there exists an initial bounded* DRg-Graph *where each edge is associated with a three-entry tuple $w = (\kappa_u, \kappa_l, \kappa_u - \kappa_l)$. As illustrated in ϵ-refining algorithm, we update the weight w of each edge with its real matching utility in the descending order of utility difference $(\kappa_u - \kappa_l)$. In Figure 7(b), the matching $(d_2, \{r_2, r_3\})$ has the maximum utility difference of 4. Thus, we first update $(13, 9, 4)$ with $\kappa(d_2, \{r_2, r_3\}) = 12$. Now, since $Score_{ub}(\mathcal{G}_B)/Score_{lb}(\mathcal{G}_B)$ = 20/16 = 1.25 > 1.2, we continue to update the matching utility. Similarly, in Figure 7(c), we update the edge with $(7, 4, 3)$ with the real utility $\kappa(d_2, \{r_1\}) = 6$. Then, we have $Score_{ub}(\mathcal{G}_B)/Score_{lb}(\mathcal{G}_B)$ = 19/18 = 1.06 < 1.2. Thus, the matching plan $P = \{(d_1, \{r_1\}), (d_2, \{r_2, r_3\})\}$ with a utility lower bound $Score_{lb}(\mathcal{G}_B) = 18$ is returned as the final answer.*

Theorem 6. *[Approximation analysis] Let $\mathcal{P}$ be the matching plan derived by Algorithm 4 and $\mathcal{P}^*$ be the optimal matching plan, then we have*

$$Score(\mathcal{P}) \geq \frac{1}{\epsilon} \cdot Score(\mathcal{P}^*). \tag{15}$$

Proof. Please refer to Appendix A.6 □

Complexity analysis. Algorithm 4 takes $O(|\mathcal{E}^+|log|\mathcal{E}^+|)$ to initialize the queue, takes $O(|\mathcal{E}^+|)$ time to calculate utility bounds, and takes $O(\psi\beta m^3)$ time to compute $\mathcal{P}$ where $O(\beta m^3)$ is the time cost of BMA algorithm except the construction of $\mathcal{G}_B$, and ψ is the probability to invoke BMA algorithm (ψ is usually very small). Assume that the threshold ϵ is satisfied in η iterations. Here, $|\mathcal{E}^+|$ is the number of edges in $\mathcal{G}_B$. Therefore, the total time complexity is $O(|\mathcal{E}^+|(log|\mathcal{E}^+| + \eta) + \psi\beta m^3)$.

6 Performance Evaluation

In this section, we experimentally evaluate the efficiency and effectiveness of our four proposed algorithms. The first is the integer liner programming approach described in Section 4.1 (referred to as ILP), the second is the bipartite matching algorithm in Algorithm 2 (referred to as BMA), the third is the fast ridesharing matching algorithm in Algorithm 3 (referred to as FMA), and the fourth is the ϵ-refining approximate algorithm in Algorithm 4 (referred to as eRA). In addition, we also evaluate the performance of our four proposed algorithms against three state-of-the-art methods [24] (referred to as SbA), [43] (referred to as PBM) and [6] (referred to as ILP-J). The performance of our algorithms is evaluated in terms of the average elapsed time, the average number of served riders (denoted as # of matchings), and the average utility score. We also conduct some additional experiments to evaluate the quality of the results.

6.1 Experimental Settings

Datasets. We evaluate the proposed algorithms on two real trajectory datasets collected from NYCTaxi[1] and DIDI[2], and a real social network collected from Gowalla[3], which have been widely used to study ridesharing problems [24]. Specifically, the DIDI dataset collects the trips in Chengdu for one month; the NYCTaxi dataset records the taxi trips in New York for one month; the Gowalla dataset contains 196,591 users and 950,327 relations, and each user is associated with an average of 8 keywords (i.e., interests). Since users' social data involves user privacy, relevant laws and regulations prohibit commercial platforms from sharing these sensitive data. Thus, it is difficult to obtain the social data for riders and drivers (i.e., relations and interests), which is a common

[1]http://www.nyc.gov.
[2]https://www.didiglobal.com.
[3]https://snap.stanford.edu/data.

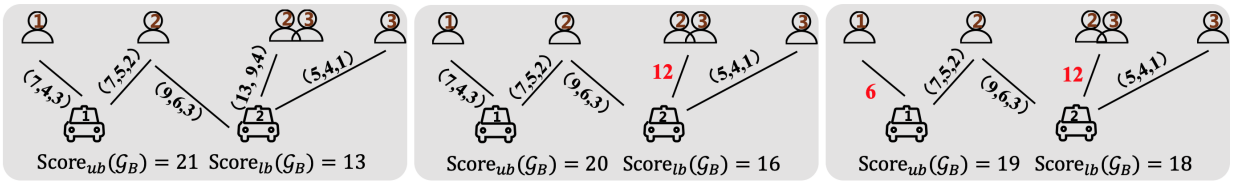


Fig. 7: An example of ridesharing matching for the ϵ-Refining Approach.

TABLE 2: Parameter settings

Parameters	*Value*	*Default*
max waiting time (mins)	*2, 3, 4, 5, 6*	*5*
tolerant detour ratio	*0.1, 0.2, 0.3, 0.4, 0.5*	*0.2*
time window (s)	*5, 10, 15, 20, 25*	*15*
number of arrival riders	*0.9K, 1.8K, 2.7K, 3.6K, 4.5K*	*2.7K*
value of α	*0.1, 0.3, 0.5, 0.7, 0.9*	*0.5*
threshold	*1.1, 1.2, 1.3, 1.4, 1.5*	*1.2*

obstacle faced by the research community. For this reason, we follow the settings of existing works [23], [24], [16], [14] to perform experiments on datasets synthesized from multiple sources of real trips, relations, and interests. We randomly select users for our experiments from the Gowalla dataset. We extract the road networks from OpenStreetMap to construct the underlying road networks for Chengdu and NewYork. Specially, the road network of Chengdu contains 36,630 intersections, 50,786 roads and the road network of NewYork contains 264,346 intersections, 366,923 roads. Then, we retrieved the riders and drivers from datasets NYCTaxi and DIDI, respectively. We map the riders and drivers to users that are randomly selected from Gowalla for our experiments. Note that in our setting, we follow the existing work [43] using the records in the peak ordering periods 7:00 am-7:30 am to evaluate the compared algorithms where there are 4,914 riders and 2,386 drivers in the DiDi dataset and 4,856 riders and 2,198 drivers in NYCTaxi dataset. Since the density of the arriving riders in this period is usually the largest in a day, it can better verify the performance of different algorithms. For the discount function $f(\Delta)$, in our experiment, we specify it as $f(\Delta) = \Delta$.

Parameters and setup. We conduct our experiments under different parameter settings to evaluate the processing performance of the proposed algorithms. The parameter settings are summarized in Table 2. All algorithms are implemented in *Java* programming language and evaluated on the *PC* with an *Intel i9-9900K @ 3.60HZ CPU* and *32GB DDR4 RAM*.

6.2 Experimental Results

6.2.1 Effect of the waiting time

Figure 8 reports the performance of all algorithms varying by different waiting time. As expected, the exact algorithms ILP, BMA and PBM achieve almost the best performance in terms of the number of served riders (# of matchings) and utility scores. However, the average running time for ILP, PBM and ILP-J is significantly higher than other algorithms and increases with the increase of waiting time. It is obvious that the longer waiting time places a larger number of riders into the search space, and this large number requires more time to find the best result. Compared with ILP, PBM and ILP-J, the algorithms FMA, eRA, and SbA reduce the running time a lot but sacrifice a little utility. While compared with BMA, FMA and eRA do not show extremely obvious advantages in running time. The reason is that although FMA and eRA can reduce the running time of matching, they need to spend time on constructing the DRg-graph. Since the running times of ILP are extremely high, it is not scalable on real-time scenario; we only report the results here and skip their evaluations in the remaining experiments.

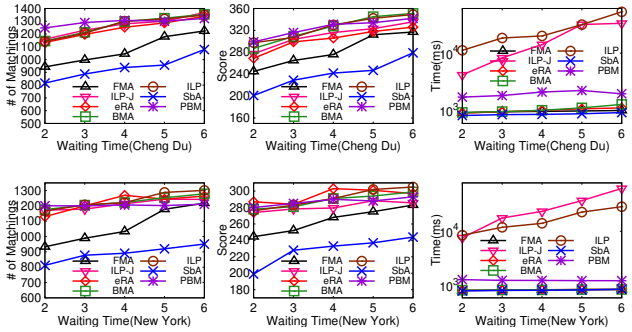


Fig. 8: Effect of the waiting time.

6.2.2 Effect of the detour ratio

In this experiment, we investigate how different values of detour ratios affect the performance of the algorithms. As shown in Figure 9, the number of served riders, the total utility, and the running time for the compared algorithms except PBM all increase when the value of the detour ratio increases. On the one hand, more riders can be served at a higher detour ratio, leading the increased total utility. On the other hand, a larger detour ratio results in more riders involved in the search space, indicating that more running time is required to find the best matching plan. However, since PBM is a packing-based approach, the increase of the tolerable detour rate allows more riders to be packed together, which reduces its running time. From the comparison of these algorithms, we can observe that for each value of the detour ratio, BMA has the largest number of served riders and the highest utility. PBM has the much higher running time than other algorithms. Our proposed method FMA and eRA run slightly faster than PBM at the expense of a slight loss in quality.

6.2.3 Effect of the time window size

In Figure 10, we report the performance of the proposed algorithm by varying time window size. Since SbA tackles the *URM* problem in a first-come-first-served approach that does not refer to time window size, we only report the performance of FMA, eRA, and BMA. Generally, a larger

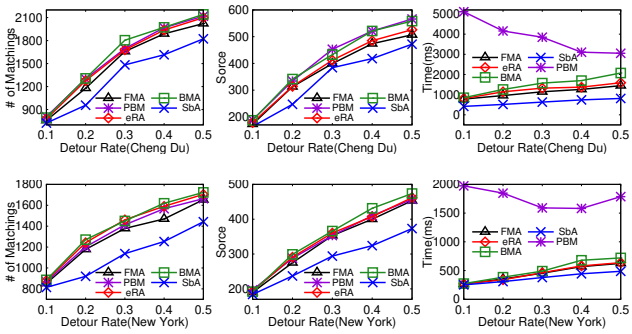


Fig. 9: Effect of the detour ratio.

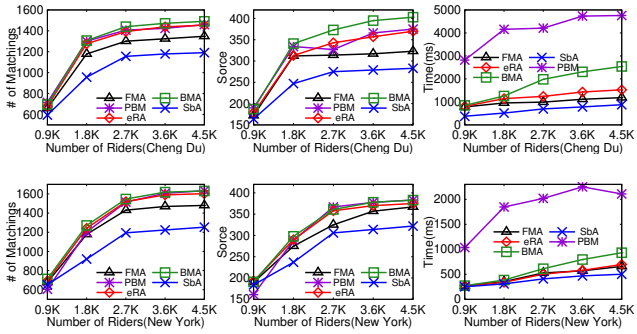


Fig. 11: Effect of the number of arriving riders.

time window size results in a larger number of riders to be matched, and that is usually used to test the algorithm scalability. This explains why the running time of all algorithms increases when the time window size is larger. From Figure 10, we can observe that the number of served riders and the total utility for all algorithms except PBM become larger. The reason is that a larger time window size helps to achieve better matchings for drivers and riders globally. However, it also offers more packing opportunities resulting in the decrease in matched riders, score and running time of PBM. Compared with BMA, FMA and eRA show a compromise performance between the quality and time cost. We also note that when the time window size is larger than 15 seconds, the number of served riders and the value of utility of FMA and eRA are close to those of BMA, which demonstrates that our proposed eRA also achieves good performance in most cases.

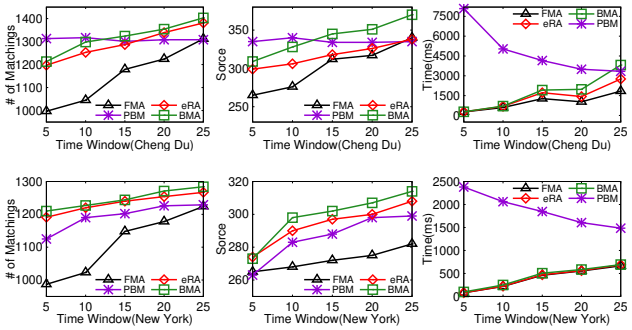


Fig. 10: Effect of the time window size.

6.2.4 Effect of the number of arriving riders

We vary the number of riders arriving from $0.9K$ to $4.5K$ to evaluate the reliability of algorithms under different request pressures. Figure 11 reports the performance of different algorithms. With the increase of the number of arriving riders, the number of served riders also increases, but there is a slowing growth when the number is set from $3.6K$ to $4.5K$. That is because the capacity of the ride offers reaches a bottleneck and barely deals with more ride requests with the current number of ride offers.

6.2.5 Effect of the value of α

In Figure 12, we report the performance of the algorithms by varying the value of α from 0.1 to 0.9. It can be seen that our proposed algorithms show advantages in effectiveness or efficiency. Compared to PBM, our proposed algorithms BMA, FMA and eRA have comparable numbers of matchings and utility scores while run much faster than PBM. In addition, since the number of drivers and riders is constant, we can observe that varying α has a very limited impact on number of matchings and running time. Different from the effects on number of matchings and running time, varying α has a great impact on the utility score of all algorithms. The reason is that although we have normalized both social comfort and price revenue into [0,1], their data distributions may be different. In this experiment, the value of social comfort is mostly close to 0 while the value of price revenue is generally close to 1. As such, when the value of α increase, i.e., the platform prefers to find the matchings with higher social comfort, then the over utility score decreases. This is also why we provide parameter α for the platform to adjust the matching results based on its business needs.

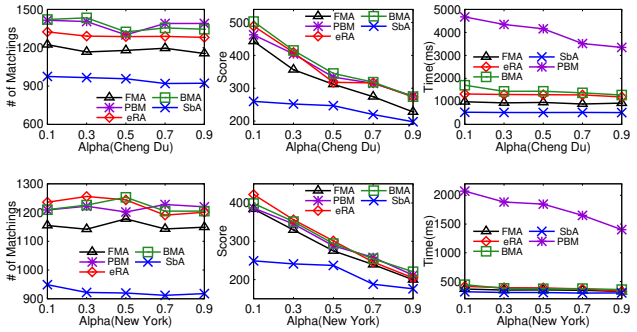


Fig. 12: Effect of the value of α.

6.2.6 Parameter sensitivity evaluation on ϵ

Since FMA, BMA, and SbA do not involve the parameter ϵ, we only report in this experiment how the threshold ϵ affects the approximate results of eRA. As mentioned in Section 5.2, the parameter ϵ is the critical factor to determine the matching quality and stop condition. Users can balance the result quality and the efficiency of the algorithm by tuning the threshold ϵ. From Figure 13, it can be seen that with the increase of the threshold from 1.1 to 1.5, in both Chengdu and New York datasets, the quality and the running time of eRA decreases. The reason is that a larger threshold ϵ allows the algorithm to reach expected results with fewer iterations. Yet the computational cost decreases at the cost of a certain quality loss.

6.2.7 Effect of the throughput

The throughput is an important metric to evaluate the processing performance of algorithms. It refers to the number of riders matched per second. In this experiment, we

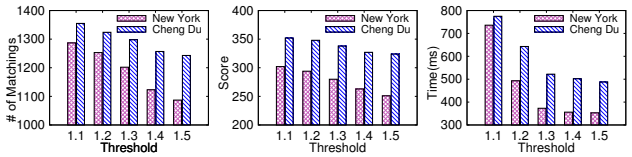


Fig. 13: Effect of threshold ϵ.

evaluate the throughput of eRA, BMA, FMA, SbA and PBM under the default values of all parameters. In Table 3, SbA shows better performance as it is the faster algorithm; FMA, eRA and BMA take the second place. However, PBM is particularly slow. The reason is that PBM spends much more time to find the best packaging scheme and matching plan.

TABLE 3: Throughput of different algorithms

Dataset	Sba	FMA	eRA	BMA	PBM
NewYork	64	59	56	52	5
Chengdu	42	24	19	17	3

6.2.8 Effect of the memory usage

Table 4 reports the memory cost (unit MB) for different algorithms under default parameter settings. It can be observed that PBM and BMA use the most memory since they need to retrieve many cases of the bipartite graph. eRA stores the upper and lower bound for achieving better efficiency. SbA takes the least memory as it match riders and drivers immediately.

TABLE 4: Memory cost of different algorithms

Dataset	Sba	FMA	eRA	BMA	PBM
New York	943	1,334	1,783	2,478	3736
Cheng Du	604	1,493	1,628	1,838	2555

7 Conclusion and Future Work

In this paper, we propose a novel URM-problem that combines social comfort and price revenue together into dynamic ridesharing. In practical applications, such considerations not only improve the comfort of ridesharing between strangers and drivers, but also ensure the interest fairness among passengers, drivers and platform. Next, with the aim of maximizing the overall utility, we propose an efficient bipartite matching algorithm based on a novel DR-Graph index that enables algorithms to tackle situations where one driver serves multiple passengers. Moreover, we present several heuristic algorithms to balance efficiency and effectiveness. Experiments on real-world datasets demonstrate that our proposed algorithms achieve the desirable ridesharing matching results and meet the efficiency requirements of real-time scenarios under the order volume of large cities such as New York and Chengdu.

As for future work, we intend to extend this work in two aspects. First, we plan to investigate the predictive-aware ridesharing that integrates real-time and predictive requests to further improve the matching quality. Second, we hope to design a more flexible ride model which, through a transfer approach, allows a rider to be cooperatively served by multiple drivers.

8 Acknowledgments

This work is supported by NSFC Grants 61972362, 61602420, and 62036010, HNSF Grant 202300410378, CPSF Grant 2018M630836, and RGC grants 12200817 and 12201615.

References

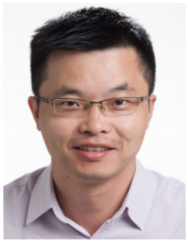
[1] Didi. https://www.didiglobal.com/.
[2] Lyft. https://www.lyft.com.
[3] Techcrunch. https://www.techcrunch.com/.
[4] Uber. https://www.uber.com.
[5] T. Abeywickrama, V. Liang, and K.-L. Tan. Optimizing bipartite matching in real-world applications by incremental cost computation. *Proc. VLDB Endow.*, 14(7):1150–1158, 2021.
[6] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc. Natl. Acad. Sci.*, 114(3):462–467, 2017.
[7] N. Armenatzoglou, S. Papadopoulos, and D. Papadias. A general framework for geo-social query processing. *Proc. VLDB Endow.*, 6(10):913–924, 2013.
[8] M. Bellmore and G. L. Nemhauser. The traveling salesman problem: A survey. *Oper. Res.*, 16(3):538–558, 1968.
[9] L. Chen, Y. Gao, Z. Liu, X. Xiao, C. S. Jensen, and Y. Zhu. Ptrider: A price-and-time-aware ridesharing system. *Proc. VLDB Endow.*, 11(12):1938–1941, 2018.
[10] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen. Price-and-time-aware dynamic ridesharing. In *Proc. IEEE Int. Conf. Data Eng.*, pages 1061–1072, 2018.
[11] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen. Price-and-time-aware dynamic ridesharing. In *Proc. IEEE Int. Conf. Data Eng.*, pages 1061–1072, 2018.
[12] Y. Chen, D. Guo, M. Xu, G. Tang, T. Zhou, and B. Ren. Pptaxi: Non-stop package delivery via multi-hop ridesharing. *IEEE Trans. on Mobile Computing*, 19(11):2684–2698, 2019.
[13] P. Cheng, C. Feng, L. Chen, and Z. Wang. A queueing-theoretic framework for vehicle dispatching in dynamic car-hailing. In *Proc. IEEE Int. Conf. Data Eng.*, pages 1622–1625, 2019.
[14] P. Cheng, H. Xin, and L. Chen. Utility-aware ridesharing on road networks. In *Proc. ACM Int. Conf. Management of Data*, pages 1197–1210, 2017.
[15] B. Cici, A. Markopoulou, E. Frías-Martínez, and N. Laoutaris. Assessing the potential of ride-sharing using mobile and social data: a tale of four cities. In *Proc. ACM Int. Conf. Ubiquitous Comp.*, pages 201–211, 2014.
[16] X. Fu, J. Huang, H. Lu, J. Xu, and Y. Li. Top-k taxi recommendation in realtime social-aware ridesharing services. In *Proc. of Int. Symp. Spatial and Temporal Databases*, pages 221–241, 2017.
[17] X. Fu, C. Zhang, H. Lu, and J. Xu. Efficient matching of offers and requests in social-aware ridesharing. *GeoInformatica*, 23(4):559–589, 2019.
[18] J. Gao, T. Wong, B. Selim, and C. Wang. Voma: A privacy-preserving matching mechanism design for community ridesharing. *IEEE Trans. Intell. Transp. Syst. (Early Access)*, 2022.
[19] S. Guo, C. Chen, J. Wang, Y. Ding, Y. Liu, X. Ke, Z. Yu, and D. Zhang. A force-directed approach to seeking route recommendation in ride-on-demand service using multi-source urban data. *IEEE Trans. on Mobile Computing*, 2020.
[20] H. W. Kuhn. The hungarian method for the assignment problem. *Nav. Res. Logist.*, 2(1-2):83–97, 1955.
[21] E. L. Lasmar, F. O. de Paula, R. L. Rosa, J. I. Abrahão, and D. Z. Rodríguez. Rsrs: Ridesharing recommendation system based on social networks to improve the user's qoe. *IEEE Trans. on Intelligent Transportation Systems*, 20(12):4728–4740, 2019.
[22] C. Levinger, N. Hazon, and A. Azaria. Human satisfaction as the ultimate goal in ridesharing. *Future Gener. Comput. Syst.*, 112:176–184, 2020.
[23] Y. Li, R. Chen, L. Chen, and J. Xu. Towards social-aware ridesharing group query services. *IEEE Trans. on Serv. Comput.*, 10(4):646–659, 2017.

[24] Y. Li, J. Wan, R. Chen, J. Xu, X. Fu, H. Gu, P. Lv, and M. Xu. Top-k vehicle matching in social ridesharing: A price-aware approach. *IEEE Trans. Knowl. Data Eng.*, 33(3):1251–1263, 2021.
[25] Z. Liu, Z. Gong, J. Li, and K. Wu. Mobility-aware dynamic taxi ridesharing. In *Proc. IEEE Int. Conf. Data Eng.*, pages 961–972, 2020.
[26] H. Ma, F. Fang, and D. C. Parkes. Spatio-temporal pricing for ridesharing platforms. *Operations Research*, 70(2):1025–1041, 2022.
[27] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Trans. on Knowl. and Data Eng.*, 2015.
[28] S. Madria, S. Yeung, and K. Ward. Ridesharing-inspired trip recommendations. In *IEEE MDM*, pages 34–39, 2018.
[29] J. Munkres. Algorithms for the assignment and transportation problems. *J. Soc. Ind. App. Math.*, 5(1):32–38, 1957.
[30] R. Pamula and R. Chakraborty. Taxi recommender system using ridesharing service. In *Proc. Int. Conf. Advanced Computing and Communication Systems*, pages 1–6, 2017.
[31] J. Pan, G. Li, and J. Hu. Ridesharing: Simulator, benchmark, and evaluation. *Proc. VLDB Endow.*, 12(10):1085–1098, 2019.
[32] D. Rheingans-Yoo, S. D. Kominers, H. Ma, and D. C. Parkes. Ridesharing with driver location preferences. In *Proc. Int. Joint Conf. Artificial Intelligence*, pages 557–564, 2019.
[33] R. L. Rosa, E. L. Lasmar Junior, and D. Zegarra Rodríguez. A recommendation system for shared-use mobility service through data extracted from online social networks. *J. of Communications Software and Systems*, 14(4):359–366, 2018.
[34] S. Saisubramanian, C. Basich, S. Zilberstein, and C. V. Goldman. Satisfying social preferences in ridesharing services. In *Proc. IEEE Int. Conf. Intelligent Transportation System*, pages 3720–3725, 2019.
[35] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong. An efficient ride-sharing framework for maximizing shared route. *IEEE Trans. Knowl. Data Eng.*, 30(2):219–233, 2018.
[36] L. Tang, Z. Liu, Y. Zhao, Z. Duan, and J. Jia. Efficient ridesharing framework for ride-matching via heterogeneous network embedding. *ACM Trans. Knowl. Discov. Data*, 14(3):27:1–27:24, 2020.
[37] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *Proc. VLDB Endow.*, 11(11):1633–1646, 2018.
[38] T. Wang, H. Luo, Z. Bao, and L. Duan. Dynamic ridesharing with minimal regret: Towards an enhanced engagement among three stakeholders. *IEEE Trans. Knowl. Data Eng. (Early Access)*, 2022.
[39] P. Welke, A. Markowetz, T. Suel, and M. Christoforaki. Three-hop distance estimation in social graphs. In *Proc. IEEE Int. Conf. Big Data*, pages 1048–1055, 2016.
[40] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li. An efficient insertion operator in dynamic ridesharing services. In *Proc. IEEE Int. Conf. Data Eng.*, pages 1022–1033, 2019.
[41] H. Yu, H. Zhang, X. Yu, X. Du, and M. Guizani. Pgride: Privacy-preserving group ridesharing matching in online ride hailing services. *IEEE Internet Things J.*, 8(7):5722–5735, 2021.
[42] W. E. Zhang, A. Shemshadi, Q. Z. Sheng, Y. Qin, X. Xu, and J. Yang. A user-oriented taxi ridesharing system with large-scale urban GPS sensor data. *IEEE Trans. Big Data*, 7(2):327–340, 2021.
[43] L. Zheng, L. Chen, and J. Ye. Order dispatch in price-aware ridesharing. *Proc. VLDB Endow.*, 11(8):853–865, 2018.

Yafei Li received the PhD degree in computer science from Hong Kong Baptist University, in 2015. He is currently a professor in the School of Computer and Artificial Intelligence, Zhengzhou University, China. His research interests span mobile and spatial data management, location-based services, and urban computing. He has authored more than 20 journal and conference papers in these areas, including IEEE TKDE,IEEE TSC, ACM TWEB, ACM TIST, PVLDB, IEEE ICDE, WWW, etc.

Huiling Li received the BEng degree in software engineering from Zhengzhou University, in 2021. He is currently working toward the MEng degree in the School of Computer and Artificial Intelligence of Zhengzhou University. His research interests include multi-agent computing, deep learning and spatiotemporal data processing.

Xin Huang received the PhD degree from the Chinese University of Hong Kong (CUHK) in 2014. He is currently an Assistant Professor at Hong Kong Baptist University. His research interests mainly focus on graph data management and mining.



Jianliang Xu received the BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, and the PhD degree in computer science from the Hong Kong University of Science and Technology. He is a professor in the Department of Computer Science, Hong Kong Baptist University. He held visiting positions with Pennsylvania State University and Fudan University. His research interests include big data management, mobile computing, and data security and privacy. He has published more than 200 technical papers in these areas. He has served as a program cochair/vice chair for a number of major international conferences including IEEE ICDCS 2012, IEEE CPSNA 2015, and APWeb-WAIM 2018. He is an associate editor of the IEEE Transactions on Knowledge and Data Engineering and the Proceedings of the VLDB Endowment 2018.

Yu Han received the BEng degree in computer science and technology from Zhengzhou University, China, in 2018. He received the MEng degree from Zhengzhou University in 2021. His research interests include multi-agent computing, deep learning and spatial-temporal data processing.

Mingliang Xu received the PhD degree from the State Key Lab of CAD&CG, Zhejiang University, China. He is a professor in the School of Computer and Artificial Intelligence of Zhengzhou University, China. His current research interests include computer graphics, multimedia and artificial intelligence. He has authored more than 60 journal and conference papers in these areas, including ACM TOG, IEEE TPAMI/TIP/TCSVT, ACM SIGGRAPH (Asia)/MM, ICCV, etc.