Top-k Vehicle Matching in Social Ridesharing

Li, Yafei; Wan, Ji; Chen, Rui; Xu, Jianliang; Fu, Xiaoyi; Gu, Hongyan; Lv, Pei; Xu, Mingliang

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI:
10.1109/TKDE.2019.2937031

Published: 01/03/2021

Document Version:
Peer reviewed version

Top-k Vehicle Matching in Social Ridesharing: A Price-aware Approach

Yafei Li, Ji Wan, Rui Chen, Jianliang Xu, Xiaoyi Fu, Hongyan Gu, Pei Lv, and Mingliang Xu

Abstract—In the past few years ridesharing has largely reshaped the transportation marketplace. It is envisioned as a promising solution to transportation-related problems in metropolitan cities, such as traffic congestion and air pollution. In the current ridesharing research, social ridesharing, which makes use of social relations among drivers and riders to address safety issues, and dynamic pricing are two active directions with important business implications. Simultaneously optimizing social cohesion and revenue is vital to a commercial ridesharing platform's sustainable development, which, however, has not been previously studied. In this paper, we first present a new pricing scheme that better incentivizes drivers and riders to participate in ridesharing, and then propose a novel type of Price-aware Top-k Matching (PTkM) queries which retrieve the top-k vehicles for a rider's request by taking into account both social relations and revenue. We design an efficient algorithm with a set of powerful pruning techniques to tackle this problem. Moreover, we propose a novel index tailored to our problem to further speed up query processing. Extensive experimental results on real datasets show that our proposed algorithms achieve desirable performance for real-world deployment.

Index Terms—Price revenue, social ridesharing, location-based services, query processing, social acquaintance.

1 Introduction

In the past few years ridesharing has largely reshaped the transportation marketplace. We have witnessed the burgeoning of several major commercial ridesharing platforms, such as *Uber* [31], *Lyft* [20], and *DiDi* [12]. Revenue in the global ridesharing segment amounts to 60 billion US dollars in 2018 and is expected to reach 109 billion in 2022, showing an annual growth rate of 16.3% [2]. More broadly, ridesharing is envisioned as a promising solution to transportation-related problems in metropolitan cities, such as traffic congestion and air pollution, and hence has been attracting substantial research interest. In the current ridesharing research, *social ridesharing* and *dynamic pricing* have been two active directions with important business implications.

Safety is a top priority of every ridesharing company. Safety incidents have already shifted all major ridesharing service providers to the center of public opinion [1]. Failure to effectively address safety concerns will largely undermine users' trust in ridesharing services. The concept of social ridesharing was consequently proposed to establish trust and accountability between drivers and riders, which conduces to resolving the notable barriers including social discomfort and safety concerns when traveling with strangers. The general idea of social ridesharing is to enforce certain social relations among drivers and riders by leveraging an underlying social network. Several recent works have started to tackle the trust issue in ridesharing services [11], [18], [24]. The techniques proposed in [24] include the adoption of a reputation-based system and user profile checking via linking social networks such as Facebook. A limitation of this approach is that it requires significant involvement from participants, making it less practical for real-world services. Cici *et al.* [11] and Li *et al.* [18] suggest to enforce certain social acquaintance relations when forming a ridesharing group. However, the social constraints given in [11] and [18] are overly restrictive, which hinders their application to real-world ridesharing services.

Social concerns are just one aspect of the ridesharing problem. For commercial ridesharing platforms, pricing schemes are an equally important facet. A good pricing scheme can directly improve user engagement, better motivate drivers and increase revenue. The current pricing schemes of commercial ridesharing platforms are *static* in the sense that a rider's fare is determined in advance and cannot change during the whole trip even when the vehicle picks up new riders on the fly. Recently, several studies [5], [9] have attempted to solve the dynamic pricing optimization problem in ridesharing. Their schemes aim at maximizing a platform's revenue which is modeled as the difference between riders' fares and drivers' incomes. A driver's income is proportional to his/her travel distance, and a rider's fare is a function of the shortest distance between his/her pick-up and drop-off locations and the amount of detour. While this pricing scheme is generally insightful, we observe several counterintuitive results in practical settings. First, the scheme prefers the vehicle generating the highest revenue instead of the vehicle bringing the maximal revenue increase. This fact might, unfortunately, lower the platform's revenue. Second, since a driver's income is tied to the travel distance, it would reduce the incentive to pick up a new

- *Y. Li, J. Wan, H. Gu, P. Lv, and M. Xu are with the School of Information Engineering, Zhengzhou University, Zhengzhou, China. E-mail: {ieyfli, wanji, guhy, ielvpei, iexumingliang}@zzu.edu.cn Corresponding author: Mingliang Xu*
- *R. Chen is with the College of Computer Science and Technology, Harbin Engineering University, China. E-mail: ruichen.hkbu@gmail.com*
- *J. Xu and X. Fu are with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong SAR, China. E-mail: {xujl, xiaoyifu}@comp.hkbu.edu.hk*

rider with a trip on the driver's current schedule, which is intuitively a desirable ridesharing assignment. Similarly, a driver would not be motivated to pick up multiple riders with the same pick-up and drop-off locations as his/her income is identical to that of picking up just a single rider. Third, if a driver is not incentivized to pick up riders with a small amount of detour, riders will suffer longer waiting time, which lowers the platform's service quality. As such, we devise a new pricing scheme to address the above concerns.

While there is no doubt that social ridesharing and dynamic pricing are two desirable features that should be offered *simultaneously* in real-world ridesharing services, to the best of our knowledge, there is no prior research studying how to combine them in a practical way. In this paper, we propose a novel type of price-aware social ridesharing queries, called *Price-aware Top-k Matching (PTkM)* query, which integrates a novel pricing scheme into social ridesharing services to identify the best candidate vehicles. Specifically, a *PTkM* query takes three input parameters r, tp_r, and k, where r is a rider, tp_r is r's trip request, and k is the number of returned vehicles. The trip request tp_r can be represented as $(l_r^p, l_r^d, t_r^p, t_r^w, d_r^c)$, where l_r^p denotes the pick-up point, l_r^d the drop-off point, t_r^p the desired pick-up time, t_r^w the maximum waiting time, and d_r^c the tolerable detour rate. Given an underlying social network, a *PTkM* query returns the top-k vehicles that represent the best choices for the platform in terms of revenue and social cohesion while satisfying various constraints (e.g., t_r^w and d_r^c).

The convergence of dynamic pricing and social ridesharing brings new challenges. First, the social constraints used in prior work are overly restrictive, which will lower the possibility of forming a ridesharing group when dynamic pricing is also under consideration. Designing a flexible social model is a vital ingredient to improve the usability of ridesharing. Second, simultaneously optimizing pricing and social factors is computationally prohibitive, especially in a practical setting where thousands of vehicles need to be processed in real time. In this paper, we propose a highly efficient algorithm with a set of effective pruning techniques to retrieve the top-k matching vehicles. To further accelerate query speed, we also design two novel safe-region-based spatial index structures called *SR-QTree* and *ESR-QTree* for processing *PTkM* queries. The major contributions of this paper are summarized as follows.

- We formally define a novel type of *PTkM* queries that combines both social and pricing aspects in ridesharing to return desirable vehicles. As suggested by our user survey results, considering both social and pricing factors is of important business values for commercial ridesharing platforms.
- We propose a new pricing scheme that better motivates drivers, increases a commercial ridesharing platform's revenue, and possibly lowers riders' waiting time.
- We propose an efficient *PTMFinder* algorithm to answer *PTkM* queries. We further improve query performance by exploring a set of effective pruning techniques and an early stop condition to accomplish the searching process as quickly as possible.

TABLE 1: Summary of Notations

Notation	Definition
G_s=(U_s, E_s)	a social network with user set U_s and relation set E_s.
G_n=(L_n, E_n)	a road network with location set L_n and road set E_n.
r, d, v	a rider, driver, and vehicle, respectively.
V	the set of vehicles moving on the road network.
$\mathcal{G}_v$	a user set including the members of v.
tp_r	the trip plan of the rider r.
l_r^p, l_r^d	the pick-up and drop-off points of r's trip plan, respectively.
t_r^p, t_r^w, d_r^c	the pick-up time, maximum waiting time, maximum detour rate of r's trip request, respectively.
d_v^c	the maximum detour rate of a vehicle v.
d_v	the available detour distance of a vehicle v.
l_v^c	the current location of a vehicle v.
S_v	the schedule of a vehicle v.
$\Pi(S_v)$	the road network distance of S_v.
$\pi(\cdot, \cdot)$	the road network distance between two points.
$\pi_v(\cdot, \cdot)$	the actual travel between two points.
$f(r, S_v)$	the ridesharing fare of a rider r.
Δ_r	the actual detour distance when r shares a ride.
Δ_v	the extra detour distance for v boarding a rider.
$R(r, v)$	the revenue of v generated by boarding rider r .
$h(\cdot,\cdot)$	the number of hops between two users.
$SC(r, v)$	the social cohesion of v boarding rider r.
$Rank(r,v)$	the ranking score of v boarding rider r.
ξ	the maximum speed limit of the road network.
$\tau(l)$	the time window of a point l in S_v.
s_v	the number of available seats in v.
$\pi_{min}(\cdot,\cdot)$	the minimum distance between the points or entries.
$Rank_{ub}(r,v)$	the ranking upper bound when v carries r.

- We design a novel index structure called *SR-Qtree* and its variant *ESR-Qtree* to encode a set of provable safe regions (*i.e.*, reachable areas), time constraints, and the bounds of revenue and social cohesion. They endow *PTMFinder* with extra pruning capabilities.
- We have conducted extensive experimental evaluations using real-life datasets. The experimental results demonstrate the efficiency and effectiveness of our proposed solution.

The rest of this paper is organized as follows. Section 2 introduces our new pricing scheme and formalizes the *PTkM* query problem. Section 3 details the baseline *PTMFinder* algorithm and an advanced algorithm. Section 4 presents the designs of a basic *SR-Qtree* and the enhanced *ESR-Qtree*. We evaluate the performance of our proposed algorithms in Section 5. Section 6 reviews the related works. Finally, we conclude this paper and discuss future work in Section 7.

2 PROBLEM FORMULATION

In this section, we present some preliminaries and provide the problem statement, followed by an example to illustrate the proposed problem. Table 1 gives a summary of the notations used in this paper.

2.1 Framework

In this paper, we focus on a new ridesharing service called *Hitch* [28] which has been widely adopted in major ridesharing platforms, including *Uber* [31], *Lyft* [20], and *DiDi* [12]. In this service, there are a large number of part-time drivers who are vehicle owners willing to offer rides to riders. A vehicle's origin and destination are the same as those of the

driver. If some riders share a ride with the driver, the driver departures from his/her origin, then picks up them from their pick-up points and drops off them to their drop-off points, and finally drives to the driver's destination.

The general framework of the *PTkM* query service is illustrated in *Fig. 1*. Our proposed ridesharing system consists of three parties: (i) riders (or passengers) who want to participate in ridesharing, (ii) drivers (or private car owners) who offer ridesharing, and (iii) ridesharing service provider (RSP) (or the server) in charge of arranging ridesharing trips. The riders submit ride requests to the RSP, while the drivers send in ride offers. In other words, a ride request submitted by a rider forms a *PTkM* query; the ride offers submitted by drivers form the data space (or search space). The RSP identifies the best ride matchings by jointly considering trip matching, social cohesion, its revenue increase as well as the capacity of a vehicle. The RSP dynamically maintains all active ride offers. Once it receives a ride request from a rider, the RSP will match the most suitable top-k drivers with the highest ranking scores to the rider. The rider can then select one of the k options as his/her final choice.

We remark that this service model has been adopted by many ridesharing companies such as Didi [12] and Yidao [37], as it provides practical benefits to riders. There are a lot of factors that may affect users' ridesharing experience, for example, the actual waiting time, the actual fare, vehicles' makes, and drivers' ratings. Riders' preferences for these factors are highly personalized and are difficult for the RSP to model as a whole. Therefore, providing top-k drivers will provide riders more flexibility to choose the best driver according to their personalized preferences.

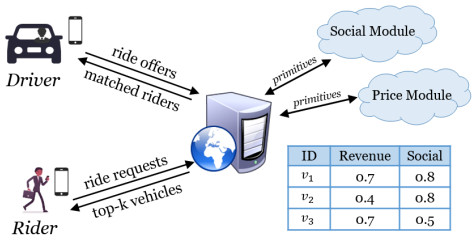


Fig. 1: A System Framework of *PTkM* Query Service

2.2 Problem Definition

A *PTkM* query is defined over a vehicle set V, a social network G_s=(U_s, E_s), and a road network G_n=(L_n, E_n). For the social network G_s, each vertex $u \in U_s$ is a user (*i.e.*, driver or rider) and each edge $e \in E_s$ denotes an acquainted relation between the two users it links. For the road network G_n, each vertex $l \in L_n$ represents an intersection and each edge $e' \in E_n$ denotes a road segment between two intersections. Each rider r issues a trip request as a tuple tp_r = (l_r^p, l_r^d, t_r^p, t_r^w, d_r^c), where l_r^p denotes the pick-up point, l_r^d the drop-off point, t_r^p the desired pick-up time, t_r^w the maximum waiting time, and d_r^c the tolerable detour rate. Each vehicle $v \in V$ is associated with four parameters: the current location l_v^c, a set of users $\mathcal{G}_v$ already onboard, the maximum detour rate d_v^c, and the number of available seats s_v. In the sequel, we use either the terms trip and ride request or driver and vehicle interchangeably. After the RSP receives a ride request, it returns the top-k most suitable vehicles by maximizing its revenue and social cohesion while satisfying various constraints.

Before giving the formal definition of a *PTkM* query, we need to present a novel pricing scheme to quantify the fare of a rider and the revenue of a vehicle, which addresses the limitations of the existing pricing scheme. The first definition we introduce is the schedule of a vehicle.

Definition 1. *(Schedule) The schedule of a given vehicle v is a sequence of location points ordered by time S_v = (l_1, l_2, $\cdots$, $l_{2|\mathcal{G}_v|}$), where $|\mathcal{G}_v|$ is the number of users in v and l_i represents the pick-up or drop-off point of a user in v. In particular, l_1 and $l_{2|\mathcal{G}_v|}$ denote the driver's origin and destination, respectively.* ■

Note that a schedule S_v of vehicle v is *valid* if it satisfies the following constraints: (i) Order-based constraint. For each rider r in v, l_r^p should precede l_r^d in S_v. (ii) Waiting time constraint. For each rider r in v, v can reach r's pick-up point l_r^p in the time duration from t_r^p to $t_r^p + t_r^w$; (iii) Detour constraint. The actual travel distance $\pi_v(l_r^p, l_r^d)$ in S_v for each rider r in v should be no more than r's longest tolerable travel distance $(1 + d_r^c) \cdot \pi(l_r^p, l_r^d)$, where $\pi(l_r^p, l_r^d)$ is the shortest path distance between points l_r^p and l_r^d. (iv) The actual detour distance of v in the whole trip should not exceed v's maximum detour distance. Furthermore, we say a schedule S_v is *optimal* if the total travel distance $\Pi(S_v)$ of S_v is minimum. Next, we give the definitions of the fare of a rider and the revenue of a vehicle.

Definition 2. *(Fare of a Rider) Given a vehicle v, the fare of a rider r in $\mathcal{G}_v$ is*

$$f(r, S_v) = \pi(l_r^p, l_r^d) \cdot \theta(\Delta_r) \cdot \kappa, \tag{1}$$

where $\theta(\cdot)$ is a monotone function of price discount with value range $[0, 1]$, κ is the per kilometer price, and $\Delta_r = \frac{\pi_v(l_r^p, l_r^d)}{\pi(l_r^p, l_r^d)}$ - 1 denotes the actual detour ratio when r shares a ride in v. The bigger Δ_r is, the more discount r receives. ■

Without loss of generality, our pricing model is introduced based on a static road network. However, in some cases, it is also desirable to calculate the fare in terms of travel time. Note that our proposed algorithms can also be extended to time-aware road networks where the travel time can be approximated by the travel distance and the travel speed [9], [21], [23], [30] or estimated in real time using travel time prediction techniques [29], [33].

Definition 3. *(Revenue of a Vehicle) Consider a vehicle (or driver) v and a rider r. Let S'_v and S_v be the schedules with and without boarding r, respectively. The revenue of v generated by boarding r is*

$$R(r, v) = \sum_{r' \in (r \cup \mathcal{G}_v - v)} f(r', S'_v) - \sum_{r' \in (\mathcal{G}_v - v)} f(r', S_v) - \tau \cdot \Delta_v. \tag{2}$$

Here $\sum_{r' \in (r \cup \mathcal{G}_v - v)} f(r', S'_v) - \sum_{r' \in (\mathcal{G}_v - v)} f(r', S_v)$ gives the revenue increment when v chooses to board r, Δ_v is the extra detour distance for v to pick up and drop off r, τ is a fixed per kilometer maintenance cost, and $\tau \cdot \Delta_v$ is the extra maintenance cost for v due to picking up and dropping off the new rider r. ■

The RSP gets a cut with a fixed rate in the form of royalties on each vehicle's revenue. In this case, the platform's revenue is proportional to all vehicles' total revenue. For

this reason, in the rest of this paper we do not explicitly distinguish the platform's revenue from vehicles' revenue.

We argue that Definitions 2 and 3 provide a more reasonable pricing scheme benefiting all three parties in the ridesharing service (i.e., the service platform, riders and drivers) from at least three aspects: (i) Unlike the static pricing schemes used by existing commercial platforms, our pricing scheme is *dynamic* so that riders can gain an extra fare discount when there is a detour to pick up and drop off other riders; (ii) Drivers are better incentivized to pick up riders with a small amount of detour, which is a natural ridesharing assignment. This may also lead to shorter waiting time for riders; (iii) Compared with the existing dynamic pricing schemes [5], [9], for a ride request our scheme selects the vehicle that brings maximal *revenue increase* instead of maximal revenue, which is guaranteed by Theorem 1. For ease of explanation, we also illustrate the difference between these two pricing schemes in Example 1. As such, our pricing scheme can well address the counterintuitive results of the existing dynamic schemes discussed in Section 1.

Theorem 1. *Consider a vehicle set V. Let R(V) and R'(V) be the total revenues of RSP that serves a single ride request r with the goals maximum revenue increase and maximum revenue, respectively. Then we have R(V)≥R'(V).*

Proof. The proof can be found in Appendix A.1.

Example 1. *Consider two vehicles v_1 and v_2. Suppose the current revenue of v_1 is 10 and the current revenue of v_2 is 3. Given a new ride request r, assume the revenue of v_1 becomes 11 when v_1 serves r and the revenue of v_2 becomes 6 when v_2 serves r. If the goal is to return the vehicle with the maximal revenue, v_1 should be the query result because v_1's revenue 11 is larger than v_2's revenue 6. The resulting total revenue of the platform is $11 + 3 = 14$. If the goal is to return the vehicle with the maximal revenue increase, v_2 should be the query result because v_2's revenue increase $6-3 = 3$ is larger than v_1's revenue increase $11-10 = 1$. In this case, the revenue of the platform is $10+6 = 16$. This example illustrates the benefit of considering revenue increase, instead of revenue itself.*

Next we discuss the social constraint used in *PTkM* queries.

Definition 4. *(Social Cohesion of a Vehicle) Given a vehicle v with the set of onboard users $\mathcal{G}_v$, the social cohesion of v resulted from adding a new rider r is*

$$SC(r, v) = \frac{|\mathcal{G}_v| \cdot (|\mathcal{G}_v| + 1)}{\sum_{r' \in r \cup \mathcal{G}_v} \sum_{r'' \in r \cup \mathcal{G}_v} h(r', r'')}, \quad (3)$$

$h(r', r'')$ is the shortest social distance between the users r' and r'' in the social network. ■

Intuitively, the social cohesion of a vehicle is the average number of hops among all pairs of users in the underlying social network, which has been widely used to measure the social cohesion of a group in many existing works [14], [25], [27], [34].

Definition 5. *(Matching) Given a set of vehicles V and a set of riders R, we call $(r, v) \in R \times V$ a matching if v has available seats and can pick up and drop off r according to r's trip constraints while not violating v's existing riders' trip constraints.* ■

To quantify the extent of a vehicle v matching the rider r, we adopt the following widely used ranking function [4], [8], [14], [19]:

$$Rank(r, v) = \alpha \cdot \frac{R(r, v)}{maxP} + (1 - \alpha) \cdot SC(r, v), \quad (4)$$

where $\alpha \in [0, 1]$ is a user-specified coefficient used to adjust the trade-off between revenue and social cohesion, *maxP* is the *maximum revenue increase* that a single ride request can bring to the RSP, which is computed from all users' historical ride requests. In the definition of ranking function *Rank(r, v)*, we measure the matching of request r and vehicle v by combing two factors: *revenue increase R(r, v)* and *average social distance SC(r, v)*. Since the value of *SC(r, v)* is in the range of [0, 1] and the value of *R(r, v)* is in the range of [0, *maxP*]. To fairly measure these two factors and keep them in the same range, we use *maxP* to normalize *R(r, v)* to the range of [0, 1].

Definition 6. *(PTkM query) Given a rider r and a set of vehicles V, a PTkM query is a tuple $q = (r, k, tp_r)$ where tp_r is a trip request issued by r that retrieves the top-k matchings $M \subset r \times V$. For each $(r, v) \in M$, it holds that $Rank(r, v) \geq Rank(r, v')$ for any $(r, v') \in r \times (V - M_V)$, where M_V is the set of vehicles in M.* ■

We illustrate the idea of *PTkM* queries using the example below.

Example 2. *Given a set of vehicles V = {v_1, v_2, v_3} moving in the road network, the table in Fig. 1 presents the revenue and social cohesion resulted from adding a rider r for each $v \in V$. Consider a top-1 PTkM query $q = (r, 1, tp_r)$. Assume that the value of α preferred by r is 0.5. We can calculate the ranking scores of v_1, v_2, and v_3: $Rank(r, v_1) = 0.5 \times 0.7 + 0.5 \times 0.8 = 0.75$, $Rank(r, v_2) = 0.5 \times 0.4 + 0.5 \times 0.8 = 0.6$, and $Rank(r, v_3) = 0.5 \times 0.7 + 0.5 \times 0.5 = 0.6$. Therefore, v_1 is returned as the final answer to q.*

In practice, the RSP receives and processes *PTkM* queries in a streaming fashion. Ideally, the RSP's objective is to maximize the sum of the ranking scores of the answers returned by all *PTkM* queries in the entire query stream. We call this problem the Total Ranking Optimization of all *PTkM* queries (*TROP*). In Theorem 2, we prove that the *TROP* problem is NP-hard by a reduction from an existing NP-hard problem, the Total Distance Optimization Taxi Ridesharing (*TDOTR*) problem [23].

Theorem 2. *The problem of TROP is NP-hard.* ■

Proof. The proof can be found in Appendix A.2.

Since the *TROP* problem is NP-hard and *PTkM* queries need to be answered in real time, we solve the *TROP* problem in a greedy manner. Concretely, we find the exact optimal solution to each *PTkM* query in real time, with the intent of finding the global optimal solution to all *PTkM* queries in the entire query stream. Nevertheless, we also provide a new theorem (Theorem 3) to analyze the competitiveness, we have proved that there exists no c-competitive ($c > 0$) deterministic online algorithm for the *PTkM* query problem. In the next two sections, we study how to efficiently process each individual *PTkM* query. This

is mainly achieved by a set of effective pruning techniques developed in Section 3 and a novel index structure proposed in Section 4.

Theorem 3. *There exists no c-competitive ($c > 0$) deterministic online algorithm for the PTkM query problem.* ■

Proof. The proof can be found in Appendix A.3.

3 Proposed Algorithms

In this section, we first present the basic algorithm named *PTMFinder* with a general framework of filtering and verification for obtaining the optimal top-k vehicles to a *PTkM* query, and then optimize *PTMFinder* with several pruning techniques and an early stop condition. Note that all algorithms proposed in this paper are based on distance-aware road networks.

3.1 PTMFinder Algorithm

Given a *PTkM* query $q = (r, k, tp_r)$, the general framework of *PTMFinder* first filters out the vehicles which cannot arrive at the pick-up point within the maximum waiting time. After that, a verification step is conducted to select the most suitable top-k vehicles according to their ranking scores.

To efficiently retrieve the top-k vehicles which can pick up the rider within the specified waiting time duration, we build a quadtree [26] (as illustrated in *Fig. 2*) to index all moving vehicles in the road network. Each level of grids in the quadtree divides the road network into a collection of contiguous entries where each entry is a square with a unique identifier. Moreover, each non-leaf entry has four child entries, and each leaf entry in the lowest level of the quadtree is associated with a set of vehicles moving in this square. The search process is similar to the typical range search on the quadtree [26].

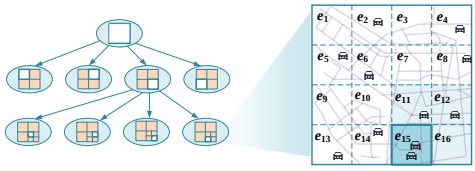


Fig. 2: An example of quadtree

Algorithm 1 gives the pseudo code of *PTMFinder*. It first creates a priority queue qu and initializes qu with the root of the quadtree *qtree*, where the priority score of qu is the minimum distance to the pick-up point l_r^p. It also initializes a vehicle set V' to store the vehicles left in the filtering step and a sorted vehicle list H to hold the final top-k vehicles in the verification step (Lines 1–3, Algorithm 1). Then Algorithm 1 iteratively processes each entry e in qu. If e is a leaf entry, it checks the distance $\pi(l_r^p, l_v^c)$ between l_r^p and each vehicle v in e. If $\pi(l_r^p, l_v^c)$ is less than or equal to $t_r^w \cdot \xi$, where ξ is the maximum speed limit of the road network[1] and l_v^c is v's current location, and the number of

1. Since a vehicle's moving speed on a road network is subject to many factors, e.g., legal speed limit of the road network, driver's experience, road congestion, weather condition, etc., which are hard to model as a whole. Thus, in this paper we follow the previous works' assumptions [22], [23], [36], which assume a vehicle's moving speed is the maximum speed limit of the road network.

Algorithm 1 PTMFinder

Input: Rider r, trip request tp_r=(l_r^p, l_r^d, t_r^p, t_r^w, d_r^c), quadtree $qtree$, road network G_n, social network G_s
Output: the top-k matched vehicles

1: Create a new priority queue qu;
2: $qu.enqueue(qtree.RootNode, 0)$;
3: Initialize an empty sorted list H and an empty set V';
4: **while** $qu \neq \emptyset$ **do**
5: Entry $e \leftarrow qu.dequeue()$;
6: **if** e is a leaf entry **then**
7: **for** each vehicle v in e **do**
8: **if** $s_v > 0$ and $\pi(l_r^p, l_v^c) \leq t_r^w \cdot \xi$ **then**
9: $V'.add(v)$;
10: **else**
11: **if** e is a non-leaf entry **then**
12: **for** each child entry e' of e **do**
13: **if** e' contains a vehicle **then**
14: **if** $\pi_{min}(l_r^p, e') \leq t_r^w \cdot \xi$ **then**
15: $qu.enqueue(e', \pi_{min}(l_r^p, e'))$;
16: **while** $V' \neq \emptyset$ **do**
17: Vehicle $v \leftarrow V'.dequeue()$;
18: $S_v \leftarrow$ compute the new schedule of v serving r;
19: **if** $S_v = \emptyset$ **then**
20: **continue**;
21: **if** $|H| < k$ **then**
22: $H.add(v)$;
23: **else**
24: Vehicle $v' \leftarrow$ the *k-th* vehicle in H;
25: **if** $Rank(r, v) > Rank(r, v')$ **then**
26: $H.remove(v')$;
27: $H.add(v)$;
28: **return** H;

$v's$ available seats s_v is more than 0, v is added into V'. If e is a non-leaf entry with at lease one vehicle, for each child entry e' of e, it computes the minimum distance between l_r^p and e', denoted by $\pi_{min}(l_r^p, e')$. If $\pi_{min}(l_r^p, e')$ is less than or equal to $t_r^w \cdot \xi$, e' is enqueued into qu. This completes the filtering step (Lines 4–15, Algorithm 1). After that, for each vehicle $v \in V'$, we use a branch and bound search algorithm to find v's optimal schedule when r boards v (Line 18, Algorithm 1). If $|H| < k$, v is added into H. Otherwise, if the ranking score *Rank(r, v)* is larger than the ranking score of the k-th entry v' in H, we remove v' and insert v to the proper position in H (Lines 21–27, Algorithm 1). Then, H is returned as the final top-k vehicles (Line 28, Algorithm 1).

Complexity. Algorithm 1 performs *filtering* and *verification* to retrieve the top-k results. The time complexity of the filtering step is $O(|V|)$, where $|V|$ is the total number of vehicles. In the verification step, it takes constant time to find the optimal schedule for a vehicle because in practice the number of available seats in a vehicle is usually small (e.g., 5), and the time complexity of the verification step is $O(|H|)$, where $|H|$ is the number of remaining vehicles after the filtering step. Therefore, the time complexity of Algorithm 1 is $O(|V| + |H|)$.

Discussion. As illustrated in Algorithm 1, given a *PTkM* query, Algorithm 1 can find the top-k vehicles with respect to the ranking score correctly. However, the search process is still inefficient for two reasons: (i) Although the filtering step discards many invalid vehicles which cannot pick up the rider within its maximum waiting time, the search process needs to enumerate all possible vehicles. (ii) It is time consuming for the branch and bound search algorithm to compute the schedule with minimum travel distance by traversing all permutations of the points. To address these weaknesses, we next present several strategies to accelerate the search process.

3.2 Advanced Algorithm

In this section, we present several pruning techniques to reduce the search space and an early stop condition to terminate the search process as early as possible.

In the sequel, we further optimize Algorithm 1 from the perspective of *filtering*. For the previous filtering step, we utilize the maximum waiting time as the bound to filter out the vehicles that cannot arrive at the pick-up point in time. In fact, considering only the waiting time for filtering is not adequate, because vehicles are also subject to detour constraints which could prevent them from picking up new riders. A vehicle v's available detour distance, denoted by d_v, is dynamic and may decrease after picking up new riders. d_v can be computed as:

$$d_v = (1 + d_v^c) \cdot \pi(l_v^p, l_v^d) - \pi_v(l_v^p, l_v^d), \tag{5}$$

where l_v^p and l_v^d denote the vehicle v's origin and destination. Accordingly, we have Theorem 4.

Theorem 4. *A vehicle v is unqualified for a trip request $tp_r = (l_r^p, l_r^d, t_r^p, t_r^w, d_r^c)$ if: (i) $\pi(l_r^p, l_v^c) > t_r^w \cdot \xi$, or (ii) $\sigma_{max}(v) < \pi(l_r^p, l_r^d)$, where l_v^c is v's current location, σ_{max} is v's maximum travel distance and $\sigma_{max}(v) = \pi_v(l_v^c, l_v^d) + d_v$.* ■

Proof. The proof can be found in Appendix A.4.

Lemma 1. *An entry e of the quadtree is unqualified for a trip request $tp_r = (l_r^p, l_r^d, t_r^p, t_r^w, d_r^c)$ if: (i) $\pi_{min}(l_r^p, e) \geq t_r^w \cdot \xi$, or (ii) $\sigma_{max}(e) < \pi(l_r^p, l_r^d)$, where $\sigma_{max}(e)$ is the maximum available travel distance of the vehicles moving in e and can be computed recursively based on the equation*

$$\sigma_{max}(e) = \begin{cases} MAX_{e' \in e}(\sigma_{max}(e')) & \text{if } e \text{ is non-leaf entry} \\ MAX_{v \in e}(\pi_v(l_v^c, l_v^d) + d_v) & \text{if } e \text{ is leaf entry} \end{cases}.$$

Here, e' is a child entry of e.■

The proof of Lemma 1 is similar to that of Theorem 4 and hence is omitted here. Intuitively, even if a vehicle v's current location l_v^c is in the circle $\odot(l_r^p, t_r^w \cdot \xi)$ centering at l_r^p with radius $t_r^w \cdot \xi$, it should still be pruned from the search space if the current available travel distance $\sigma_{max}(v)$ of v is less than $\pi(l_r^p, l_r^d)$. The observation gives more pruning power than the previous method. Lemma 1 achieves the pruning power in a similar way to Theorem 4. The only difference is that Lemma 1 allows to prune the unqualified vehicles from higher grid levels (*i.e.*, non-leaf entries), which further enhances the pruning capability. However, the whole processing still requires checking all the vehicles and cannot terminate early enough. Therefore, we propose an early stop pruning to further accelerate the search speed. We derive an upper bound of the vehicle's revenue increase for taking a new rider in Theorem 5. The intuition is that if the new rider's pick-up and drop-off points are both in the vehicle's current schedule, the new rider will bring the maximum revenue to the vehicle.

Theorem 5. *The upper bound of a vehicle v's revenue increase, denoted by $R_{ub}(r, v)$, for taking a new rider r is*

$$R_{ub}(r, v) = \pi_v(l_v^c, l_v^d) \cdot \kappa. \blacksquare \tag{6}$$

Proof. The proof can be found in Appendix A.5.

Algorithm 2 Advanced PTMFinder

Input: Rider r, trip request tp_r=$(l_r^p, l_r^d, t_r^p, t_r^w, d_r^c)$, quadtree $qtree$, road network G_n, social network G_s
Output: the top-k matched vehicles

```
1: Create two new priority queues qu_1, qu_2;
2: qu_1.enqueue(qtree.RootNode, 0);
3: Initialize an empty sorted list H;
4: while qu_1 ≠ ∅ do
5:     Entry e ← qu_1.dequeue();
6:     if e is a leaf entry then
7:         for each vehicle v in e do
8:             if s_v > 0 and π(l_r^p, v) ≤ t_r^w · ξ then
9:                 if σ_max(v) ≥ π(l_r^p, l_r^d) then
10:                    qu_2.enqueue(v, Rank_ub(r, v));
11:    else
12:        if e is a non-leaf entry then
13:            for each child entry e' of e do
14:                if e' contains a vehicle then
15:                    if π_min(l_r^p, e') ≤ t_r^w · ξ then
16:                        if σ_max(e) ≥ π(l_r^p, l_r^d) then
17:                            qu_1.enqueue(e', π_min(l_r^p, e'));
18: while qu_2 ≠ ∅ do
19:    Vehicle v ← qu_2.dequeue();
20:    S_v ← compute the new schedule of v serving r;
21:    if S_v = ∅ then
22:        continue;
23:    if |H| < k then
24:        H.add(v);
25:    else
26:        Vehicle v' ← the k-th vehicle in H;
27:        if Rank(r, v') ≥ Rank_ub(r, v) then
28:            return H;
29:        else
30:            if Rank(r, v) > Rank(r, v') then
31:                H.remove(v');
32:                H.add(v);
33: return H;
```

To keep a tight ranking upper bound, we next discuss how to derive the upper bound of the social cohesion of a vehicle v in Theorem 6.

Theorem 6. *The upper bound of the social cohesion of a vehicle v, denoted by $SC_{ub}(r \cup \mathcal{G}_v)$ of v, for taking a new rider r is as*

$$SC_{ub}(r, v) = \frac{|\mathcal{G}_v| \cdot (|\mathcal{G}_v| + 1)}{\sum_{r_i \in \mathcal{G}_v} \sum_{r_j \in \mathcal{G}_v} h(r_i, r_j) + 2 \cdot |\mathcal{G}_v|}. \blacksquare \tag{7}$$

Proof. The proof can be found in Appendix A.6.

Combining Theorems 5 and 6, we can derive the upper bound of the ranking score for v taking a given rider r.

Lemma 2. *The upper bound of the ranking score $Rank_{ub}(r, v)$ for a vehicle v taking a rider r is*

$$Rank_{ub}(r, v) = \alpha \cdot \frac{\pi_v(l_v^c, l_v^d) \cdot \kappa}{maxP} + (1 - \alpha) \cdot \frac{|\mathcal{G}_v| \cdot (|\mathcal{G}_v| + 1)}{\sum_{r_i \in \mathcal{G}_v} \sum_{r_j \in \mathcal{G}_v} h(r_i, r_j) + 2 \cdot |\mathcal{G}_v|}. \blacksquare$$

Lemma 2 is a natural result due to Theorems 5 and 6. We omit the proof here. Based on Lemma 2, we can easily derive Lemma 3 that provides extra pruning capability for ceasing the search process early enough.

Lemma 3. *Given a sorted vehicle list L_v and a new rider r, each vehicle $v \in L_v$ is sorted in descending order according to its ranking upper bound $Rank_{ub}(r, v)$. The first k vehicles in L_v are the top-k ranked vehicles if $Rank(r, v_k) \geq Rank_{ub}(r, v_{k+1})$.*

Algorithm 2 shows the pseudo code of the advanced *PTMFinder* algorithm, which integrates the pruning techniques discussed above with the baseline *PTMFinder*. In the

beginning, we create an additional priority queue qu_2 to store the vehicles retained in the filtering step in descending order of ranking upper bound (Line 1, Algorithm 2). Besides the pruning techniques presented in Algorithm 1, for each vehicle v or each entry e in qu_1, we also check if v or e satisfies the conditions presented in Theorem 4. If yes, they will be enqueued into qu_2 or qu_1 (Lines 9–10, 16–17, Algorithm 2). In the verification step, we add an additional verification to implement the pruning strategy based on Lemmas 2 and 3. If the ranking score of the current k-th vehicle in H is more than the ranking upper bound of the next vehicle to process, the search process will end and H is returned as the final top-k results (Lines 27-28, Algorithm 2). Otherwise, we repeat the process until the top-k vehicles are found.

Complexity. Following a similar analysis to that of Algorithm 1, we can learn that the time complexity of Algorithm 2 is $O(|V|+|H'|)$, where $|H'|$ is the number of retained vehicles. Due to the efficient pruning techniques and the early stop condition, $|H'|$ is normally much smaller than $|H|$ in Algorithm 1. It follows that Algorithm 2 is much more efficient than Algorithm 1.

Discussion. While the advanced *PTMFinder* has employed several pruning techniques and an early stop condition to optimize the filtering and verification procedures, it is still possible to further improve the filtering mechanism. In the next section, we show how to make use of a novel hybrid index named *SR-QTree* and its variant *ESR-QTree* to further improve the query performance.

4 Hybrid Index Based Approach

Index is a widely used technique to optimize query performance. Recently, several techniques [5], [9], [22] have been introduced to process real-time ridesharing. However, none of them can be adopted directly to solve our problem. In this section, we design a novel *Safe-Region Quadtree (SR-QTree)* index to further prune the search space and accelerate query processing. In what follows, we first describe the concept of *Safe Region* and then give details of the *SR-QTree*, followed by a variant called *Enhanced SR-QTree (ESR-QTree)*, and the corresponding query processing.

4.1 Safe Region (SR)

The search module in the proposed *PTkM* service aims at retrieving a small set of vehicles which can pick up and drop off riders in time based on their trip requests. It is easy to observe a necessary condition for a vehicle's valid schedule: the maximum travel distance of a vehicle should satisfy its maximum detour constraint. Inspired by this observation, we propose an efficient index structure called *Safe Region (SR)* to filter the candidate vehicles. Intuitively, the *SR* of a vehicle v consists of a set of leaf entries that are reachable as per v's schedule S_v. We formalize the concept of *SR* below.

Definition 7. *(SR of a Vehicle) The SR of a vehicle v is a set of leaf entries in the quadtree which overlaps any circle with the center being a road intersection in the path of the vehicle's schedule S_v and the radius being the upper bound of v's possible moving distance due to the detour constraint.* ■

Fig. 3 illustrates the *SR* of a vehicle v with the schedule $S_v = (l_v^c, l_r^p, l_r^d, l_v^d)$ in a simple road network, where the red crosses denote pick-up/drop-off points in S_v, and the round points denote the road intersections. The safe region SR_v contains the shaded leaf entries overlapping the circles.

Next, we discuss how to calculate SR_v of a vehicle v. The first step is to identify all road intersections in the path of S_v. Once we have the intersections, the problem boils down to calculating the circles' radius ψ and finding the overlapping leaf entries. Lemma 4 gives the formula to calculate ψ.

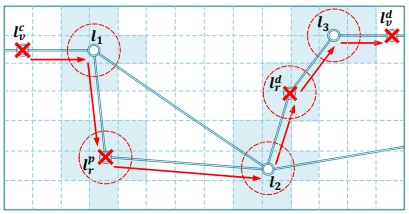


Fig. 3: An example of safe region

Lemma 4. *For a vehicle v, the upper bound of the radius ψ of a circle at a road intersection is*

$$\psi = \frac{\pi_v(l_v^c, l_v^d) + d_v}{2}, \tag{8}$$

where l_v^c is v's current location, l_v^d is v's final destination, and d_v is v's available detour distance. ■

Proof. The proof can be found in Appendix A.7.

To retrieve the leaf entries overlapping a circle, we need to calculate the distance between the entry containing the circle's center point and its nearby entries. It is time consuming to frequently execute such calculations on the fly, which is not applicable to real-time scenarios. To this end, we propose to utilize a matrix to store the pre-computed shortest distance between any two entries in each level of the quadtree. Using this distance matrix can quickly filter out the vehicles which violate the waiting time constraint. The other advantage of maintaining such a matrix is that the shortest distance between any two points (*e.g.*, the intersections of the road network) can be mapped into that of two entries, we can quickly get a distance lower bound of any two points, which is essential to reduce the cost of many pruning techniques presented in this paper.

Lemma 5. *Given a vehicle v and a new rider r, the vehicle v cannot be a candidate to the final answer if the new rider's pick-up point l_r^p and drop-off point l_r^d are not both in SR_v.* ■

It is obvious that the vehicle v cannot be a candidate vehicle to the rider r if v's reachable area cannot contain r's pick-up and drop-off points. Lemma 5 tells that only the vehicles whose safe region contains the rider's pick-up point and drop-off point would be the candidates to the final answer. Next, we present how to build the *SR-QTree* based on safe region concept.

4.2 SR-QTree

Using the safe region of a vehicle, we can quickly determine whether it is a candidate for an arriving request. However, we still need to explicitly check all vehicles to generate the top-k results. Our insight is that it is possible to devise a novel hybrid index *SR-QTree* that encodes aggregate safe

region information into a quadtree structure so as to prune unqualified vehicles *in a batch manner*. Specifically, each node of an *SR-QTree* has four types of aggregate information: an *SR* which is the union of all its child nodes' *SRs*, the upper bound of the maximum revenue increase, the upper bound of the maximum social cohesion, and the upper bound of the maximum travel distance, which describe the properties of the group of vehicles moving inside the node. A sample *SR-QTree* is given in the dashed rectangle in *Fig. 4*.

We take a bottom-up approach to efficiently generate the *SR* of an entry in the *SR-QTree*. Specifically, the *SR* of a leaf node is the union of the *SRs* of the vehicles inside it. For a non-leaf node n, let $n_1, n_2, \cdots, n_m$ be the child nodes of n. The *SR* of n is $SR_n = \bigcup_{i=1}^{i=m} SR_{n_i}$.

Lemma 6. *Given a PTkM query $q = (r, k, tp_r)$, all vehicles rooted at n cannot be candidates to the final answer to q if any of the pick-up point l_r^p and the drop-off point l_r^d are not in SR_n.*

Lemma 6 can be easily proved and hence is omitted here. Lemma 6 provides a way to effectively prune invalid vehicles on the high level of the *SR-QTree*. For a given node n in the *SR-QTree*, if any of the pick-up and drop-off points is not in n's *SR*, then all the vehicles moving in n cannot be the candidates to the final answer.

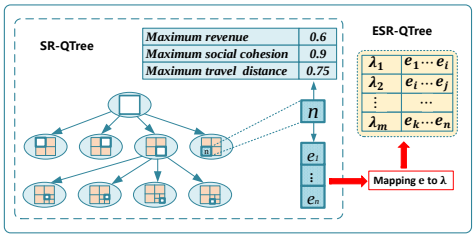


Fig. 4: An example of *SR-QTree* and *ESR-QTree*

In addition, each node n indexes two variables $rMax$ and $sMax$ to track the maximum revenue and maximum social cohesion for the vehicles inside n, respectively. We adopt a bottom-up approach to calculate these two variables for all nodes. The advantage of having these two variables is that, we can easily derive the ranking upper bound of the vehicles moving in the node n by the equation $Rank_{ub}(r, e) = \alpha \cdot rMax + (1 - \alpha) \cdot sMax$. Given a *PTkM* query, if we find the ranking score of the *k-th* vehicle in the priority queue is larger than the next processing node's ranking upper bound, this node and the inner moving vehicles can be pruned from the search space. Compared with the basic *PTMFinder* algorithm introduced in Section 3.1 and the advanced algorithm discussed in Section 3.2, *SR-Qtree* enables us to prune entries from its high levels, which can significantly accelerate the search process.

4.3 ESR-Qtree

While an *SR-Qtree* provides effective pruning capability based on vehicles' available detour distances, it can be further improved by taking into consideration vehicles' time constraints. For example, assume that a rider r's pick-up and drop-off points are in a vehicle v's safe region, but the time when v arrives at r's pick-up point or drop-off point is not in the time windows of r's pick-up point or drop-off point, respectively. In this case, v is not a candidate to the final answer, but cannot be pruned by using an *SR-Qtree*. Based on this observation, we present an *Enhanced SR-QTree* structure called *ESR-QTree*, which makes use of the time dimension to group the leaf entries in an node's SR so as to further improve the pruning capability.

To build an *ESR-QTree*, we first divide a day into multiple non-overlapping time bins $\Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_m\}$, where m is the number of time bins. For example, when $m = 24$, each bin represents an hour. Recall that the *SR* of a node n in an *SR-QTree* consists of a set of leaf entries in the quadtree. In an *ESR-QTree*, we map each leaf entry e in n's *SR*, denoted by SR_n into a time bin λ_i and store it into a bucket B_{λ_i}. The mapping function is determined by the time window $\tau(e)$ (or $\tau(l)$) when vehicles whose safe region contains e (or l) enter and leave e (or l). If the time window $\tau(e)$ of a leaf entry e spans multiple time bins, e will be stored in all their buckets. Therefore, the next key step is to calculate the time window $\tau(e)$ for each leaf entry e in SR_n.

The time window $\tau(e)$ of a leaf entry e is determined by the vehicles whose *SR* contains e. Let this set of vehicles be V_e. For each $v \in V_e$, its time window w.r.t. e is denoted by $\tau(v, e) = [t_{v,e}^+, t_{v,e}^-]$, where $t_{v,e}^+$ is the earliest time when v can arrive at e and $t_{v,e}^-$ is the latest time when v can leave e. $t_{v,e}^+$ and $t_{v,e}^-$ can be calculated as follows:

$$t_{v,e}^+ = t_c + \frac{\pi_{min}(e, e_c)}{\xi} \tag{9}$$

$$t_{v,e}^- = t_c + \frac{\sigma_{max}(v) - \pi_{min}(e, e_d)}{\xi} \tag{10}$$

where t_c is the current time, $\pi_{min}(e, e_c)$ is the minimum travel distance between the leaf entry e and v's current locating leaf entry e_c, and $\sigma_{max}(v) - \pi_{min}(e, e_d)$ is the maximum travel distance that v moves to the leaf entry e_d where v's destination locates. After we have calculated $\tau(v, e)$ for all $v \in V_e$, the time window $\tau(e)$ can be derived from $\tau(e) = \bigcup_{v \in V_e} \tau(v, e)$ (i.e., merge overlapping time windows). e is then assigned to all time bins which overlap $\tau(e)$. We can efficiently construct an *ESR-QTree* from an *SR-QTree* in a bottom-up manner.

Theorem 7 states how to make use of an *ESR-QTree* to perform the filtering process.

Theorem 7. *Consider a trip request tp_r and a node n in* ESR-QTree *with the safe region SR_n. n contains candidate vehicles to the final answer only if there exist two leaf entries e_1 and e_2 in SR_n satisfying: (i) $l_r^p \in e_1$ and $l_r^d \in e_2$, and (ii) $\tau(l_r^p) \cap \tau(e_1) \neq \emptyset$ and $\tau(l_r^d) \cap \tau(e_2) \neq \emptyset$.* ■

The proof of Theorem 7 is simple and thus omitted here. We can easily integrate the *ESR-QTree* structure with Algorithm 2 and then make use of Theorem 7 to filter out unqualified vehicles. Example 3 illustrates the pruning process based on Theorem 7.

Example 3. *Consider the safe region SR_n of node n in the* ESR-QTree *shown in Fig. 5. Consider a coming trip request in which the pick-up point l_r^p is in the leaf entry e_1, the drop-off point l_r^d is in the leaf entry e_3, and the time windows of l_r^p and l_r^d are $\tau(l_r^p) = [10:00, 10:05]$ and $\tau(l_r^d) = [10:30, 10:35]$, respectively. Since e_1 and e_3 are both in SR_n, we can learn that there may exist vehicles in node n that can arrive at e_1 and e_3*

without violating their available detour distances. Next we make use of their time constraints for further pruning. Since $\tau(l_r^p) \subset \lambda_1$ *and* $\tau(l_r^d) \subset \lambda_4$*, we check whether* $e_1 \in B_{\lambda_1}$ *and* $e_3 \in B_{\lambda_4}$*. Since* $e_3 \notin B_{\lambda_4}$*. Therefore, all vehicles in node* n *will be pruned from the search space because none of them can pick up or drop off the rider in time.*

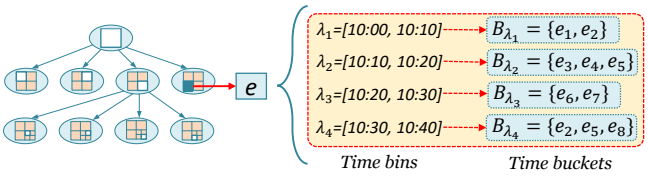


Fig. 5: An example of pruning using the *ESR-QTree*

Index Update Next, we consider the issue of index update. The *SR-QTree* and *ESR-QTree* will be updated in the following situations: (i) a vehicle enters or leaves the system; (ii) a new rider is assigned to a vehicle; (iii) a vehicle moves from one leaf entry to another. As for the method of updating the *SR-QTree* and *ESR-QTree*, we first recompute the aggregate information (*e.g.*, the safe region, the upper bound of the maximum revenue increase, the upper bound of the maximum social cohesion, and the upper bound of the maximum travel distance) of a vehicle and then update them in the *SR-QTree* and *ESR-QTree* in a bottom-up manner following the approach presented in Section 4.2.

5 Performance Evaluation

In this section, we experimentally evaluate the performance of three algorithms. The first is the *Baseline PTMFinder* presented in Section 3.1 (referred to as *Baseline*), the second is the *Advanced PTMFinder* consisting of *Baseline* and the pruning strategies presented in Section 3.2 (referred to as *Advanced*), and the last is the *SRBased PTMFinder* integrating *Advanced* with the hybrid index *ESR-QTree* presented in Section 4.3 (referred to as *ESRBased*). We evaluate the overall query performance using the real road networks of *NewYork* and *ChengDu* in terms of the average elapsed time, which have been widely used in previous studies. We also conduct several additional experiments to evaluate the quality of query results.

5.1 Experimental Settings

We evaluate the proposed algorithms over two real trajectory datasets collected from *NYCTaxi*[2] and *DiDi*[3], which have been widely used in ridesharing related research. The dataset *NYCTaxi* consists of one month's taxi trajectories in New York. We adopt the method proposed in [35] to detect the pick-up and drop-off points of the trajectories. The dataset *DIDI* contains one month's ride orders placed in Chengdu and was released by *DiDi*. We consider ride orders as riders' trip requests. Table 2 summarizes the properties of these two datasets. Since users' social data is highly sensitive and needs to be handled with care, there are different legal regulations in place that prevent commercial ridesharing platforms like *DiDi* from sharing users' social data. This is a common obstacle faced by the research community. To this end, we follow the previous works' design [6], [7], [10], [18] to conduct experiments on datasets synthesized from real trips and social relationships obtained from different sources. We select a set of users from the real social network *Gowalla*[4] as drivers and riders. We make use of the road network information from OpenStreetMap[5] to construct the underlying road networks. In the road networks, we randomly release a certain number of vehicles and randomly generate queries for the experiments. The query parameters are summarized in Table 3, where k is the number of vehicles returned by a query and α is the trade-off between revenue and social cohesion in the ranking function. For the default price discount function $\theta(\Delta_r)$, per kilometer price κ, and per kilemeter maintenance cost τ used in the experiments, we configure them as: $\theta(\Delta_r) = 1 - 0.5 \times \Delta_r$ and $\kappa = 2$.

TABLE 2: Dataset Properties

Items	*NewYork*	*ChengDu*
Total # of users	*196,591*	*196,591*
Total # of relations	*950,327*	*950,327*
Total # of vehicles	*20,000*	*15,000*
Total # of intersections	*264,346*	*36,630*
Total # of roads	*366,923*	*50,786*

All the algorithms are implemented in Java programming language and tested on a machine with an *Intel i7-7700 @ 3.60GHz* CPU and *16GB* RAM.

TABLE 3: Parameter Settings

Parameters	*Value*	*Default*
k	*1, 2, 3, 4, 5*	*3*
α	*0.1, 0.3, 0.5, 0.7, 0.9*	*0.5*
number of vehicles	*0.5k, 1.5k, 5k, 10k, 15k*	*5k*
maximum waiting time (minute)	*2, 5, 8, 10, 15*	*10*
maximum detour rate	*0,1, 0.3, 0.5, 0.7, 0.9*	*0.5*
side length of leaf entry (meter)	*0.05k, 0.1k, 1k, 5k, 10k*	*0.1k*
capacity of vehicle	*1, 2, 3, 4, 5*	*4*

5.2 Experimental Results

In this section, we report the query processing performance of the three algorithms in terms of *the average elapsed time* and *the quality of query results*.

Effect of k. In the first set of experiments, we investigate how different values of the parameter k affect the performance of different algorithms. In general, the query time increases when the value of k increases. This is because a larger number of returned vehicles requires more time to maintain the set of top-k candidates. Compared with the other two algorithms, *SRBased* is less sensitive to the increase of k. Even when the number is 5, the performance of *SRBased* is still reasonably good, demonstrating the effectiveness of the *ESR-QTree*.

Effect of the number of vehicles. We next evaluate the performance by varying the number of vehicles in the road network. From Fig. 7, it can be observed that with the increase of the number of vehicles the average query time also increases sharply. Compared with *Baseline*, *Advanced*

2. http://www.nyc.gov
3. https://gaia.didichuxing.com
4. https://snap.stanford.edu/data
5. https://www.openstreetmap.org

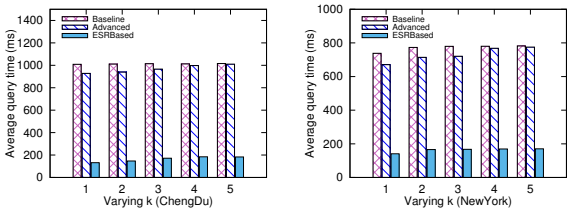


Fig. 6: Running time vs. k

achieves better performance because it benefits from the early stop condition proposed in Section 3. Due to the efficient pruning ability of the index structure *ESR-QTree*, *SRBased* dramatically decreases the average query time, with up to 5X performance improvement. It confirms our theoretical analysis that the *ESR-QTree* index structure can efficiently prune a large number of vehicles which cannot pick up and drop off riders in time, leading to a smaller search space.

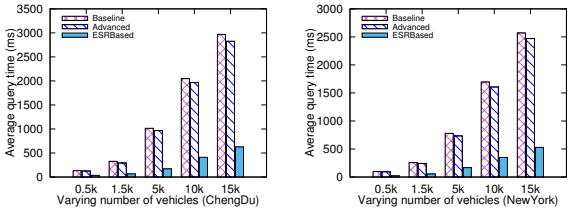


Fig. 7: Running time vs. the number of vehicles

Effect of the maximum detour rate. We show the effect of the maximum detour in Fig. 8. Intuitively, a larger maximum detour allows more vehicles to be potentially qualified and thus requires more time to compute the schedule, revenue, and social cohesion. This is why the average elapsed time increases as the maximum detour increases. Due to the pruning capability of safe region, *SRBased* achieves the best performance among all three algorithms.

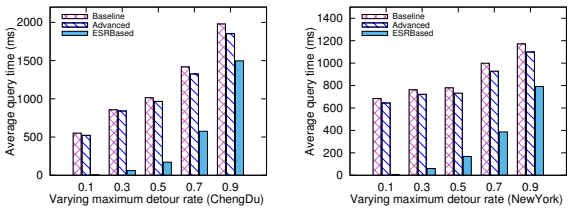


Fig. 8: Running time vs. the maximum detour rate

Effect of the side length of leaf entry. The side length of leaf entries in the quadtree plays an important role in query runtime. As such, we study query performance by varying the side length of leaf entries in Fig. 9 to draw insights on selecting proper leaf entry side lengths. It can be observed that when the side length is more than $0.1k$ meters and less than $1k$ meters, the performance of all three algorithms is relatively good and stable. When the side length gets larger or smaller (e.g., greater than $5k$ meters or less than $0.1k$ meters), the performance of all algorithms significantly deteriorates. The reason is that a too large or too small entry side length generally weakens the pruning ability in the first filtering step of all three algorithms. In particular, when the entry side length is large, it also increases the size of safe region, which makes *ESR-QTree* less effective, and therefore the performance of *ESRBased* also declines quickly.

In the experiments, we also investigate the performance of *SR-QTree* and *ESR-QTree* by varying the side length of leaf entries, since the side length of leaf entries in *quadtree-based* indexes plays an important role in query performance. It can be seen that the *ESR-QTree* based approach (*ESRBased*) consistently has better performance than the *SR-QTree* based approach (*SRBased*). This is because taking into consideration the time constraints to group the leaf entries in a node's safe region can enhance the pruning capability of *ESR-QTree*. It can filter out the vehicles whose arrival times at a rider's pick-up and drop-off points cannot match the time constraints of the ride request. Hence, *ESR-QTree* provides more effective pruning capability than *SR-QTree*. The better pruning capability of *ESR-QTree* is at the cost of additional storage overhead. For example, the sizes of *ESR-QTree* and *SR-QTree* on the *New York* dataset are 362 MB and 259 MB, respectively.

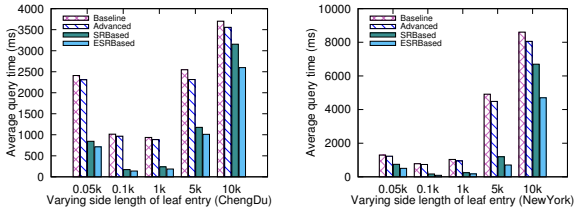


Fig. 9: Running time vs. the size of leaf entry

Quality of the results of different queries. Next, we compare the quality of the results returned by *PTkM* queries with that of the price-aware ridesharing queries (referred to as *PR*) [5] and that of the nearest neighbor queries (referred to as *NN*) used by Uber [5] in terms of average social cohesion and total revenue. In the experiments, we report the experimental results of ride matching for the entire day range, where top-3 *PTkM* queries with default settings are used. As shown in Fig. 10(a) and 10(a), *PTkM* queries outperform *PR* queries and *NN* queries in terms of both metrics. Please note that *PTkM* queries consider both social cohesion and revenue, *PR* queries focus on revenue optimization, and *NN* queries only consider schedule matching. It is expected to observe that *PTkM* queries achieve better average social cohesion. *PTkM* queries also achieve better total revenue because assigning a ride request to the vehicle with the maximum revenue increase indeed brings an RSP more revenue. Thus, we deem *PTkM* queries can provide a better ridesharing service in practical use. In addition, we examine the average fare of ride requests in Fig. 10(c). The average fare of *PTkM* queries is also less than that of *PR* queries and that of *NN* queries, bringing real benefits to riders.

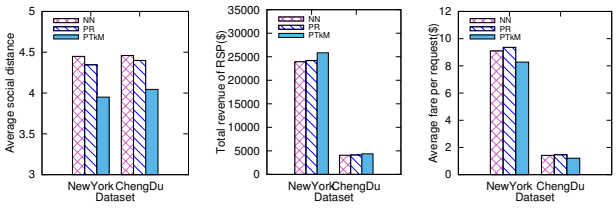


Fig. 10: Quality of different queries' results

Update cost of Safe Region. Finally, as the *Safe Region* is the core component of *SR-QTree* and *ESR-QTree*, which requires frequent update operations, we conduct experiments

to evaluate the cost of updating safe region. As shown in Fig. 11, we can observe that the update cost is less than *0.1ms* when the entry side length is greater than or equal to *1km*. This suggests that our proposed method is highly efficient and its pruning capability well justifies the slight update cost.

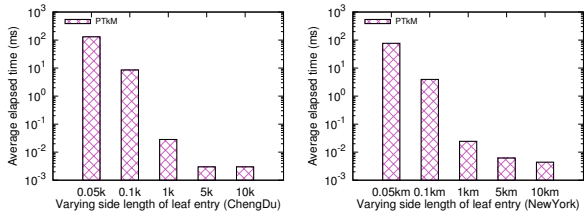


Fig. 11: Update performance of safe region

Moreover, we have performed additional experiments to evaluate the query performance under different values of waiting time, α, and capacities of vehicles. We have also conducted a questionnaire on the major concerns of riders in ridesharing, whose results well justify the need of *PTkM* queries. Due to the space limitation, we present them in Appendixes B.1, B.2, B.3, and B.4.

6 Related Works

To the best of our knowledge, there are no prior studies that simultaneously consider social ridesharing and dynamic pricing. In what follows, we mainly review studies on relevant fields: dynamic ridesharing and personalized ridesharing.

6.1 Dynamic Ridesharing

Dynamic ridesharing is a more practical ridesharing model, which brings many advantages over static ridesharing, especially in terms of the flexibility in satisfying users' needs. In dynamic ridesharing systems, riders and drivers continuously enter and leave the system, where their requests and offers are matched in real time or on a short notice. A recent survey on the optimization techniques for centralized dynamic ridesharing can be found in [3]. Various optimization objectives (e.g., minimizing system-wide vehicle miles or travel time) and spatial-temporal constraints (with desired departure/arrival time or spatial proximity requirements) have been considered. The latest work [17] considers a centralized real-time ridesharing problem with service guarantee. Tong et al. [30] propose a novel unified formulation of route planning with a well-defined parameterized objective function that can flexibly resolve multi-objective route planning problems (e.g., dynamic ridesharing for shared mobility). Several novel kinetic-tree-based algorithms are proposed to better suit dynamic request scheduling and on-the-fly route adjustment. The drawback of centralized ridesharing is the lack of scalability, especially when ridesharing requests come in a large volume. To address this issue, distributed ridesharing solutions have been developed [13], [38]. d'Orey et al. [13] propose a dynamic taxi-sharing algorithm based on peer-to-peer communications and distributed coordination. Zhao et al. [38] present a distributed ridesharing service based on a new geometry matching algorithm to shorten the waiting time for passengers and avoid traffic jams. However, all these works only consider participants' itineraries and time schedule constraints in ridesharing assignments. They cannot be applied to social-aware or price-aware ridesharing assignment problem, which involves more complex constraints.

6.2 Personalized Ridesharing

Very recently, several works have been presented to tackle the personalized issues such as privacy, utility, activity, and trust in ridesharing services [10], [15], [16], [32]. These works have analyzed the benefits of the proposed personalized solutions from different perspectives.

Agatz et al. [24] propose to adopt a reputation-based system and check user profiles via linking social networks such as Facebook. This approach, however, requires significant involvement from participants. Cici et al. [11] suggest to group participants who are friends or friends of friends when assessing the potential benefits of ridesharing. However, such simple social constraints would be either too restricted or too relaxed to be practical for real-life ridesharing systems. Wang et al. [32] present an activity-based ridesharing which employs time geography to build a choice set of alternative destinations and aim at retrieving an optimal matching fitting best into ridesharing partners' schedules. Cheng et al. [10] formulate the problem of utility-aware ridesharing on road networks (URR) with the goal of providing the optimal rider schedules for vehicles to maximize the overall utility subject to spatial-temporal and capacity constraints. They also propose an efficient algorithm to achieve a minimum increase in travel cost without reordering the existing schedule of a vehicle. From the point of view of privacy, Hallgren et al. [16] propose a model named PrivatePool for privacy-preserving ridesharing. They develop a decentralized architecture to provide strong security and privacy guarantees without sacrificing the usability of ridesharing services.

In contrast, we present a new ridesharing query service with a focus on social ridesharing and dynamic pricing, which have important business implications.

7 Conclusion and Future Work

In this paper, we have proposed a novel type of *PTkM* queries that tackles the convergence of social ridesharing and dynamic pricing. A *PTkM* query aims at retrieving the *top-k* most suitable vehicles in which the driver and riders benefit the most not only in revenue/fare but also social cohesion. We have designed a line of efficient algorithms to answer *PTkM* queries, and also conduct an extensive empirical study on real datasets to demonstrate that the proposed algorithms achieve desirable query performance.

As for the future work, we plan to extend this work in two aspects. First, we intend to investigate more personalized ride requests to make our proposed *PTkM* queries more practical. Second, we attempt to design more effective algorithms with tightly provable approximation bounds to answer *PTkM* queries efficiently.

Acknowledgments

This work is supported by NSFC Grants 61602420, 61972362, 61772474, 61672469, 61822701, CPSF Grant 2018M630836, Henan Key R&D Grant 192102310476, and RGC Grants

12200817 and 12201615. Part of Yafei Li's work was done when he visited the Database Research Group with Hong Kong Baptist University.

REFERENCES

[1] Didi's safety problem is every ride-share company's problem. https://money.cnn.com/2018/08/31/technology/didi-uber-lyft-safety/index.html. Accessed: 2018-09-16.

[2] Ride sharing. https://www.statista.com/outlook/368/100/ride-sharing/worldwide. Accessed: 2018-09-15.

[3] N. Agatza, M. Savelsbergh, and X. Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

[4] N. Armenatzoglou, S. Papadopoulos, and D. Papadias. A general framework for geo-social query processing. *Proc. VLDB Endowment*, 6(10):913–924, 2013.

[5] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *Proc. ACM Int'l Conf. Advances in Geographic Information Systems*, 2016.

[6] F. Bistaffa, A. Farinelli, G. Chalkiadakis, and S. D. Ramchurn. Cooperative game-theoretic approach to the social ridesharing problem. *Artificial Intelligence*, 246:86–117, 2017.

[7] F. Bistaffa, A. Farinelli, and S. D. Ramchurn. Sharing rides with friends: A coalition formation algorithm for ridesharing. In *Proc. the 29th AAAI Conf. on Artificial Intelligence*, pages 608–614, 2015.

[8] L. Chen, Y. Li, J. Xu, and C. S. Jensen. Towards why-not spatial keyword top-k queries: A direction-aware approach. *IEEE Trans. on Knowledge and Data Engineering*, 30(4):796–809, 2018.

[9] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen. Price-and-time-aware dynamic ridesharing. In *Proc. IEEE Int'l Conf. on Data Engineering*, pages 1061–1072, 2018.

[10] P. Cheng, H. Xin, and L. Chen. Utility-aware ridesharing on road networks. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pages 1197–1210, 2017.

[11] B. Cici, A. Markopoulou, E. Frias-Martinez, and N. Laoutaris. Assessing the potential of ride-sharing using mobile and social data: a tale of four cities. In *Proc. ACM Int'l Joint Conf. on Pervasive and Ubiquitous Computing*, pages 201–211, 2014.

[12] D. Company. http://www.didiglobal.com.

[13] P. M. D'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *Proc. Int'l IEEE Conf. on Intelligent Transportation Systems*, pages 140–146, 2012.

[14] X. Fu, J. Huang, H. Lu, J. Xu, and Y. Li. Top-k taxi recommendation in realtime social-aware ridesharing services. In *Proc. Int'l Symposium on Spatial and Temporal Databases*, pages 221–241, 2017.

[15] X. Fu, C. Zhang, H. Lu, and J. Xu. Efficient matching of offers and requests in social-aware ridesharing. In *Proc. 19th IEEE Int'l Conf. on Mobile Data Management*, pages 197–206, 2018.

[16] P. Hallgren, C. Orlandi, and A. Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *Proc. Computer Security Foundations Symposium*, pages 276–291, 2017.

[17] Y. Huang, R. Jin, F. Bastani, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *Proc. VLDB Endowment*, 7(14):2017–2028, 2013.

[18] Y. Li, R. Chen, L. Chen, and J. Xu. Towards social-aware rideshare group query services. *IEEE Trans. on Services Computing*, 10(4):646–659, 2017.

[19] Y. Li, R. Chen, J. Xu, Q. Huang, H. Hu, and B. Choi. Geo-social k-cover group queries for collaborative spatial computing. *IEEE Trans. on Knowledge and Data Engineering*, 27(10):2729–2742, 2015.

[20] Lyft. https://www.lyft.com.

[21] S. Ma and O. Wolfson. Analysis and evaluation of the slugging form of ridesharing. In *Proc. ACM Int'l Conf. Advances in Geographic Information Systems*, pages 64–73, 2013.

[22] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *Proc. IEEE Int'l Conf. on Data Engineering*, pages 410–421, 2013.

[23] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Trans. on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[24] M. S. N. Agatz, A. Erera and X. Wang. Sustainable passenger transportation: Dynamic ride-sharing. In *Proc. Erasmus Research Inst. of Management*, 2009.

[25] J. Niu, J. Fan, L. Wang, and M. Stojinenovic. K-hop centrality metric for identifying influential spreaders in dynamic large-scale social networks. In *Proc. Global Communications Conf.*, pages 2954–2959, 2015.

[26] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[27] M. Schaal, O. John, and B. Smyth. *An Analysis of Topical Proximity in the Twitter Social Graph*. Springer Berlin Heidelberg, 2012.

[28] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong. An efficient ride-sharing framework for maximizing shared route. *IEEE Trans. on Knowledge and Data Engineering*, 30(2):219–233, 2018.

[29] D. Tiesyte and C. S. Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *Proc. 16th ACM SIGSPATIAL Int'l Symp. on Advances in Geographic Information Systems*, page 14, 2008.

[30] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *Proc. VLDB Endowment*, 11(11):1633–1646, 2018.

[31] Uber. https://www.uber.com.

[32] Y. Wang, R. Kutadinata, and S. Winter. Activity-based ridesharing: Increasing flexibility by time geography. In *Proc. ACM Int'l Conf. Advances in Geographic Information Systems*, 2016.

[33] Y. Wang, Y. Zheng, and Y. Xue. Travel time estimation of a path using sparse trajectories. In *Proc. 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 25–34, 2014.

[34] P. Welke, A. Markowetz, T. Suel, and M. Christoforaki. Three-hop distance estimation in social graphs. In *Proc. IEEE Int'l Conf. on Big Data*, pages 1048–1055, 2017.

[35] M. Xu, H. Wang, S. Chu, Y. Gan, X. Jiang, Y. Li, and B. Zhou. Traffic simulation and visual verification in smog. *ACM Trans. on Intelligent Systems and Technology*, 10(1):3:1–3:17, 2019.

[36] 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'l Conf. on Data Engineering*, pages 1022–1033, 2019.

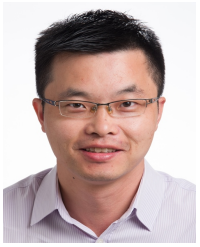
[37] Yidao. https://www.yongche.com/.

[38] W. Zhao, Y. Qin, D. Yang, L. Zhang, and W. Zhu. Social group architecture based distributed ride-sharing service in vanet. *Int'l Journal of Distributed Sensor Networks*, 2014(1):1–8, 2014.

Yafei Li received the PhD degree in computer science from Hong Kong Baptist University, in 2015. He is currently an associate professor in the School of Information Engineering, Zhengzhou University, China. His research interests span data management and location-based services. 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.

Ji Wan received the BEng degree in computer science and technology from Zhengzhou University, China, in 2015. He is currently working toward the MEng degree at the School of Information Engineering, Zhengzhou University. His research interests include temporal and spatial data management, location-based services, and urban computing.

Rui Chen received the Ph.D. degree in Computer Science from Concordia University. He is a professor in the College of Computer Science and Technology, Harbin Engineering University. His research interests include machine learning, data privacy and databases. He has published more than 40 technical papers in top venues, including CSUR, VLDBJ, PVLDB, IEEE TKDE, IEEE TDSC, ACM KDD, ACM CCS and IEEE ICDE, and won CIKM 2015 Best Paper Runner Up, the Best Papers of ICDE 2016 and the Best Papers of ICDM 2018. He has served as program committee member for leading conferences, including ACM KDD, IEEE ICDM, and ACM CIKM, and as reviewer for numerous flagship journals, including VLDBJ, PVLDB, IEEE TKDE, IEEE TDSC, IEEE TIFS, and IEEE TOPS.



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 150 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.

Xiaoyi Fu received the BEng degree in computer science and engineering from Zhejiang University. She is currently working toward the PhD degree at the Department of Computer Science, Hong Kong Baptist University, Hong Kong. She is a member of the Database Research Group with Hong Kong Baptist University. Her research interest includes query processing in spatio-temperal databases, location-based services, and ridesharing.

Hongyan Gu received the BEng degree in computer science and technology from Zhengzhou University, China, in 2016. She is currently working toward the MEng degree at the School of Information Engineering, Zhengzhou University. Her research interests include spatio and temperal databases, location-based services, and ridesharing.

Pei Lv received the PhD in 2013 from the State Key Lab of CAD&CG, Zhejiang University, China. He is an associate professor in School of Information Engineering, Zhengzhou University, China. His research interests include video analysis and crowd simulation. He has authored more than 20 journal and conference papers in these areas, including IEEE TIP, IEEE TCSVT, ACM MM, etc.

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 Information Engineering 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.

APPENDIX A
PROOFS

In this appendix, we provide the proofs of all theorems and lemmas in the paper.

A.1 Proof of Theorem 1

Proof. Let $\varepsilon(v)$ and $\varepsilon'(v)$ be the revenues of $v \in V$ serving r or not, respectively. Assume v' is the vehicle with the maximum revenue increase and v'' is the vehicle with the maximum revenue in V. Then, we have $R(V) = \varepsilon'(v') + \varepsilon(v'') + \sum_{v \in V - \{v', v''\}} \varepsilon(v)$ and $R'(V) = \varepsilon'(v'') + \varepsilon(v') + \sum_{v \in V - \{v', v'\}} \varepsilon(v)$. As v' is the vehicle in V with the maximum revenue increase, we have $\varepsilon'(v') - \varepsilon(v') \geq \varepsilon'(v'') - \varepsilon(v'')$. Accordingly, $R(V) - R'(V) = \varepsilon'(v') - \varepsilon(v') - (\varepsilon'(v'') - \varepsilon(v'')) \geq 0$. Hence, $R(V) \geq R'(V)$.

A.2 Proof of Theorem 2

Proof. We establish the hardness by a reduction from an existing NP-hard problem, namely the *Total Distance Optimization Taxi Ridesharing (TDOTR)* problem [22]. An instance of the *TDOTR* problem consists of a set of taxis $T = \{tx_1, tx_2, \cdots, tx_n\}$ and a set of ridesharing queries $Q = \{q_1, q_2, \cdots, q_m\}$. The optimization problem is to decide if we can find a solution that each ride request $q_i \in Q$ is assigned a unique taxi in $tx_i \in T$ and the total travel distance of all taxis for all requests is minimized.

Given an instance of *TDOTR*, we construct an instance of the *TROP* problem on a set of trip requests TR and a set of vehicles V. Each trip request $tp_r \in TR$ corresponds to a query $q_i \in Q$, and each vehicle $v_i \in V$ corresponds to a taxi $tx_i \in T$. We consider the restricted case of the *TROP* problem with the settings $\theta(\Delta_r) = 1$, $\Delta_v = 0$, and $\alpha = 1$. It can be seen that there exists a solution to the *TDOTR* problem if and only if there exists a solution to the *TROP* problem (i.e., each ride request is assigned a vehicle such that the total revenue of all vehicles is maximized).

Suppose we have a polynomial-time algorithm A that returns an answer to the *TROP* problem. If each trip request in TR is assigned a vehicle in V and the total revenue of all vehicles in V is maximized, then each query in Q is assigned a taxi in T and the total travel distance of all taxis in T is minimized. This implies that a polynomial-time algorithm to the *TDOTR* problem is found, leading to a contradiction. Therefore, there does not exist a polynomial-time algorithm A for the *TROP* problem. ■

A.3 Proof of Theorem 3

Proof. We establish the proof of Theorem 3 by contradiction. Suppose there exists a c-competitive deterministic online algorithm A for the *PTkM* query problem, which means, for every input ride request, the running result of A is at most c times worse than the optimal result. To prove Theorem 3, we only need to show an input ride request for which A cannot provide a result that is at most c times worse than the optimal result.

We assume there exists an adversary who knows every decision and all input ride requests of A. For ease of explanation, given a vehicle v and two riders r_1 and r_2, we assume: (1) there exist a vehicle v locating at point (0, 0), a ride request of r_1 with pick-up time t = 0 and pick-up point (τ, 0), and a ride request of r_2 with pick-up time t = 0 and pick-up point (-τ, 0); (2) each ride request has a maximum waiting time τ. Then, there are three options for A to make a decision at time t = 0: (1) moving towards point (τ, 0) to pick up r_1; (2) moving towards point (-τ, 0) to pick up r_2; (3) staying idle. If option (1) is selected, at time t = 1, the adversary can generate n more ride requests with pick-up point (-τ-1, 0) and drop-off point same as that of r_2. Furthermore, assume that the driver of v, the rider r_2, and the riders of the n new ride requests are all friends with each other. In this scenario, the global optimal solution should pick up r_2 first and then complete the n new ride requests. However, A can finish just the ride request of r_1. Similar analysis can be made if option (2) or (3) is selected. When more ride requests are processed by A in a similar situation, the adversary can make the result of A unboundedly worse than the global optimal result, leading to a contradiction to our assumption. Thus, the proof is completed.

A.4 Proof of Theorem 4

Proof. The distance that v can move during the waiting time t_r^w is at most $t_r^w \cdot \xi$, and, therefore, if the distance between v's current location l_v^c and the pick-up point l_r^p is greater than $t_r^w \cdot \xi$, v cannot be qualified. This explains condition (i). If the remaining maximum travel distance of v is less than the trip distance of r, it is obvious that v cannot drop off r at the drop-off point l_r^d in time even if v can pick up r without violating the waiting time constraint. This justifies condition (ii). ■

A.5 Proof of Theorem 5

Proof. According to Definitions 2 and 3, the new rider brings the maximum revenue only when the new rider's pick-up and drop-off points are vehicle v's current location and destination, respectively. In this situation, the vehicle does not incur any extra travel distance or maintenance cost to pick up the new rider, the new rider shares a ride with vehicle v as long as possible (*i.e.*, $\pi_v(l_v^c, l_v^d)$). The fare of the new rider fully contributes to the vehicle's revenue, and therefore the theorem holds. ■

A.6 Proof of Theorem 6

Proof. By Definition 3, the social cohesion of v can be computed via the average number of hops between user pairs in v. When the new rider r connects all the users in $\mathcal{G}_v$, the social cohesion of v is maximum. We thus have:

$$
\begin{aligned}
SC(r, v) &= \frac{|\mathcal{G}_v| \cdot (|\mathcal{G}_v| + 1)}{\sum_{r_i \in r \cup \mathcal{G}_v} \sum_{r_j \in r \cup \mathcal{G}_v} h(r_i, r_j)} \\
&= \frac{|\mathcal{G}_v| \cdot (|\mathcal{G}_v| + 1)}{\sum_{r_i \in \mathcal{G}_v} \sum_{r_j \in \mathcal{G}_v} h(r_i, r_j) + 2 \cdot \sum_{r_i \in \mathcal{G}_v} h(r, r_i)} \\
&\leq \frac{|\mathcal{G}_v| \cdot (|\mathcal{G}_v| + 1)}{\sum_{r_i \in \mathcal{G}_v} \sum_{r_j \in \mathcal{G}_v} h(r_i, r_j) + 2 \cdot |\mathcal{G}_v|} = SC_{ub}(r, v). \blacksquare
\end{aligned}
$$

A.7 Proof of Lemma 4

Proof. We prove Lemma 4 by contradiction. Consider a vehicle v with the schedule $S_v = (l_v^c, l_r^p, l_r^d, l_v^d)$. Assume for the sake of contradiction that there exists a reachable point p outside v's safe region. We have the rerouted trip schedule $S'_v = \{l_v^c, l_r^p, p, l_r^d, l_v^d\}$, which is valid for v. We have $\Pi_v(S'_v) \leq \Pi_v(S_v) + d_v$. Accordingly, $\pi(l_v^c, l_r^p) + \pi(l_r^p, p) + \pi(p, l_r^d) + \pi(l_r^d, l_v^d) \leq \pi(l_v^c, l_r^p) + \pi(l_r^p, l_r^d) + \pi(l_r^d, l_v^d) + d_v \Rightarrow \pi(l_r^p, p) + \pi(p, l_r^d) \leq \pi(l_r^p, l_r^d) + d_v$. As the point p is not in v's safe region, it means the distance between the circle's center point and p exceeds ψ. We thus have $\pi(l_r^p, p) + \pi(p, l_r^d) > 2\psi$. According to Equation 8, $\pi_v(l_v^c, l_v^d) + d_v = 2\psi$. Then we have $\pi(l_r^p, p) + \pi(p, l_r^d) > \pi_v(l_v^c, l_v^d) + d_v$. This contradicts the assumption that there exists a point p which is reachable for v but not in v's safe region. ■

APPENDIX B
SUPPLEMENTARY EXPERIMENTS

In this appendix, we provide some supplementary experiments to further evaluate the query performance.

B.1 Effect of the waiting time

In Fig. 12, we investigate the query performance by varying the maximum waiting time. It can be observed that the average elapsed time of *Baseline* and *Advanced* increases rapidly as the maximum waiting time increases, while the performance of *SRBased* is relatively stable. The average elapsed time increases because a larger maximum waiting time allows more vehicles to reach pick-up points within the waiting time period, leading to a larger search space. In this case, considering the safe region structure can effectively speed up query processing.

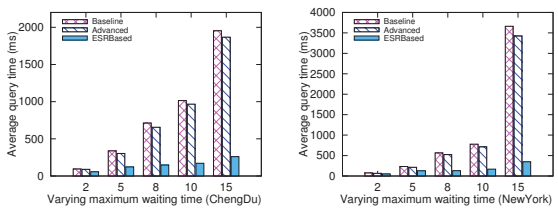


Fig. 12: Running time vs. the waiting time

B.2 Effect of α

In this set of experiments, we study the query performance of the algorithms with respect to varying α values. Recall that α represents the trade-off between social cohesion and revenue. From Fig. 13, we can observe that all three algorithms perform relatively stable under different α values and that under all different α values *SRBased* consistently outperforms the other two algorithms. This again justifies the pruning capability of safe region.

B.3 Effect of the capacity of vehicles

Fig. 14 presents the performance of the three algorithms when varying vehicles' capacity. As expected, compared with *Baseline* and *Advance*, *SRBased* achieves the best performance in all cases. When the capacity is small (i.e., 1 to 3), the average elapsed time slightly increases with

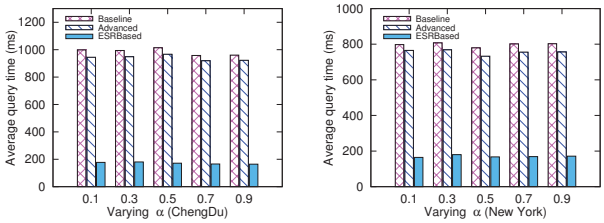


Fig. 13: Running time vs. α

the increase of capacity. This is because a larger number of available seats provides more candidate vehicles for a rider's request, leading to longer processing time. However, when the number of available seats exceeds 3, the query time flattens because a larger capability allows more riders onboard, which in turn consumes vehicles' available detour distances and reduces the search space.

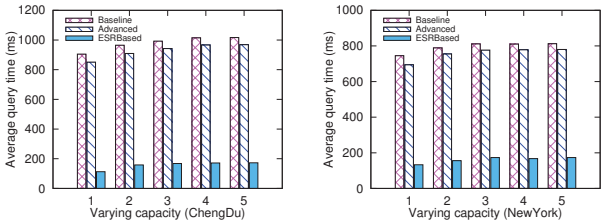


Fig. 14: Running time vs. the capacity of vehicle

B.4 Survey of riders' concerns in ridesharing

Items	Percentage
Ride safety	90.1%
Pricing scheme	76.24%
Driver's ratings	34.65%
Onboard environment	38.61%
Response time	42.57%
Actual waiting time	57.43%
Actual detour	22.77%

Fig. 15: Survey results of riders' concerns in ridesharing

We invited 100 volunteers with rich ridesharing experience to complete a user study[6] to understand the key factors affecting users' ridesharing experience from a wide range of factors such as ride safety, pricing scheme, driver's experience, onboard environment, response time, actual waiting time, and actual detour. The study results are shown in Figure 15. Ride safety and pricing scheme are the top-2 factors concerning these volunteers in ridesharing. The results well justify the design choice of considering social cohesion and the pricing scheme in *PTkM* queries.

6. https://www.wjx.cn/jq/38630773.aspx