A User-Oriented Taxi Ridesharing System with Large-Scale Urban GPS Sensor Data

Wei Emma Zhang, *Member, IEEE,* Ali Shemshadi, Quan Z. Sheng, *Member, IEEE,* Yongrui Qin, Xiujuan Xu, and Jian Yang

Abstract—Ridesharing is a challenging topic in the urban computing paradigm, which utilizes urban sensors to generate a wealth of benefits and thus is an important branch in ubiquitous computing. Traditionally, ridesharing is achieved by mainly considering the received user ridesharing requests and then returns solutions to users. However, there lack research efforts of examining user acceptance to the proposed solutions. To our knowledge, user decisions in accepting/rejecting a rideshare is one of the crucial, yet not well studied, factors in the context of dynamic ridesharing. Moreover, existing research attention is mainly paid to find the nearest taxi, whilst in reality the nearest taxi may not be the optimal answer. In this paper, we tackle the above un-addressed issues while preserving the scalability of the system. We present a scalable framework, namely TRIPS, which supports the probability of accepting each request by the companion passengers and minimizes users' efforts. In TRIPS, we propose three search techniques to increase the efficiency of the proposed ridesharing service. We also reformulate the criteria for searching and ranking ridesharing alternatives and propose indexing techniques to optimize the process. Our approach is validated using a real, large-scale dataset of 10,357 GPS-equipped taxis in the city of Beijing, China and showcases its effectiveness on the ridesharing task.

Index Terms—Big Sensory Data, Dynamic Ridesharing, Spatio-Temporal, Heterogeneous, Urban Computing and Planning.

✦

1 INTRODUCTION

WITH the growth of ubiquitous computing, nowadays we are able to derive knowledge in real-time from large and heterogeneous data collected by sensors from urban spaces [1]. Trajectory data from public transport can be gathered aiding by the sensors carried by buses, trains and taxis. These data can be utilized by many applications such as traffic control, travel planning and ridesharing. Among the number of public transport options in many urban areas, ridesharing plays an important role to relieve the problem of traffic lines which are overwhelmingly growing in our cities. In general, ridesharing provides us with various benefits such as economical (e.g., reduced total mileage and fuel consumption), environmental (e.g., less air pollution) and social (e.g., passenger waiting time) benefits [2], [3].

The ridesharing problem has been actively studied in the past few years. The problem has been considered in various forms such as Carpooling/Vanpooling, Hitchhiking, Mass Transit Systems, Dial A Ride, Web-based Shared Ride Systems (e.g., *Google ride finder* that was later replaced by *Google Transit* which in turn was integrated into *Google maps*). Dynamic ridesharing is generally described as an automated system that facilitates drivers and riders to share one-time trips close to their departure times/places and can be characterized with features like dynamic, independent, cost-sharing, non-recurring trips, prearranged and auto-

mated matching [2]. In the recent years, various research efforts have been made to facilitate ridesharing where each work focuses on particular types of results. For instance, a ridesharing application designed to minimize the effect of stochastic time frames [4] has different effects than a solution that is designed to minimize the mileage of the vehicles [3], [5].

Taxi ridesharing is a complex problem. Most often, it is not possible to find a taxi which travels exactly at the expected itineraries and schedules. This leads to the problem of searching for the most suitable taxis for dynamic sharing requests. Generally, it requires to find nearby taxis by extending the search areas around the origin and the destination points. Existing solutions on ridesharing typically exploit an *Incremental Search* (IS) strategy in which the search area gradually increases until a compromise match is found [3], [5]. A *Decremental Search* (DS) approach has been proposed to find the margin of the search space first, and then reduces the search area to find a satisfying taxi [6]. Decremental search increases the performance of search to some extent. However, these search approaches only consider the nearest taxi as the matching taxi, but do not take human factors, e.g., the acceptance probability of the companion passengers into consideration. In reality, each ridesharing request needs to be evaluated by the companion passengers and can be either accepted or rejected. Thus, the nearest taxi will not necessarily be the best choice. If the request is rejected by the companion passenger who is already booked or on the taxi, incremental and decremental search approaches have to start again to search for the next available alternative, which are not efficient. Therefore, a search approach that can consider the human acceptance probability would be a better and more practical solution.

Regarding the factors considered in ridesharing, some

- W. E. Zhang, Q. Z. Sheng, J. Yang are with the Department of Computing, Macquarie University, NSW 2109.

- A. Shemshadi is with the Complexica, Adelaide, SA 5022.

- Y. Qin is with the School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield HD1 3DH, United Kingdom.

  X. Xu is with School of Software Technology, Dalian University of Technology, Dalian, Liaoning, 116620, P.R. China

Fig. 1: (a) Rideshare requests on a companion passenger's mobile device; (b) Alternatives found by a ridesharing application ordered by distance to the user on a seeker passenger's mobile device; and (c) Alternatives ordered by the possibility of being accepted by companion passengers on a seeker passenger's mobile device

existing applications are designed to focus on only one factor (i.e., the distance) [3], [7], while some other works address a set of different factors [8], e.g., the time windows for multiple requests. Some works optimise the ridesharing benefits through maximising the number of participants [2]. However, due to the complexity of the human decision making, rare work addresses the human factors in ridesharing [9]. Thus further advances are required to maximize user participations. It requires considering related constraints such as schedules and preferences [10], [11]. In our work, we consider the acceptance probability of the companion passengers in the ridesharing process.

To further discuss the human factor in ridesharing, we describe the scenarios of ridesharing in more detail. There are three types of different users, who are involved in a typical dynamic ridesharing scenario. This includes *seeker users* who have not secured a taxi, *companion users* who are already on the taxi or scheduled a trip with a taxi which will take place shortly, and finally the *taxi driver*. For the sake of simplicity, we exclude taxi drivers from our study and only focus on the mutual interactions of the first two types of users. Typically, a seeker user uses a ridesharing application to get a shared ride with a number of companion passengers or passengers who may have already booked the taxi [3]. The companion passengers may accept or reject the ridesharing request based on their desired criteria. We particularly consider three scenarios in dynamic ridesharing, which are related to the outcome of the decisions of companion travelers. Those scenarios are illustrated in Figure 1 and explained in the following.

Scenario One. Figure 1(a) shows the list of ridesharing requests on the screen of the mobile device of a companion passenger. Based on the savings from each request, the user may pick the second request and reject the first as the second option provides better savings. However, in many cases, if the amount of savings is very low compared to the total cost of the private ride, the companion passenger may reject all requests and accept none of them.

Scenario Two. Figure 1(b) shows an example of the results of an existing ridesharing application on the seeker passenger's mobile device. The application orders the results from the seeker user's pickup point. As shown, it is not far from reality if several attempts made by the seeker user get

declined due to the preferences of the companion passengers. Only after several attempts the user have successfully found a shared ride with $taxi_8$ while the taxi is not very far from the first (nearest) taxi. If the users of the ridesharing application need to make numerous retries to get a taxi, they may cease using the application due to the hassle it takes.

Scenario Three. Along with the second scenario, we suggest to develop a new application by considering the probability of accepting the user's ridesharing requests. As shown in Figure 1(c), this time the list is ordered by a score that indicates the probability of a request acceptance. As a result, the $taxi_8$ which has the most opportunity and savings, will be on top of the list.

Based on the above observations in the scenarios and dissuasions, in this paper, we propose the TRIPS framework that maximizes the real-world savings from dynamic ridesharing by combining two important parameters: vehicle mileage (i.e., distance) and users' acceptance. Unlike current works, which suppose that the ridesharing request gets accepted and select the nearest taxi, we consider the probability of rejection in TRIPS and propose a new search approach for this purpose. To measure the probability, in this study, we focus on the economical criterion, which is the most common one, as a main factor that affects companion users decisions. We rely on the *mutual benefit principle*, which is a basic concept that demands almost anyone who is participating in the ridesharing process, mainly is looking for financial benefits. For the search step in TRIPS, by extending the decremental search idea, we propose a *Fixed Search* (FS) approach, which runs only one time per each user query. We further optimize our search using an indexing approach, producing the *Index Powered Fixed Search* (IPFS) approach. For the ranking step in TRIPS, we propose a novel ranking algorithm that exploits probabilistic partial orders. Finally, the result set will be sorted by the combination of extra mileage and acceptance probability. TRIPS uses a *search-once-and-rank* strategy rather than *extending* [5] or *shrinking* [6] the search area in each step. Moreover, TRIPS provides a layered architecture with modules to provide support for handling the uncertainty of end users' decisions. The main contributions of our work are as follows:

- We propose a novel scalable approach which improves the query results considering the uncertainty

in the decisions made by companion passengers. We propose a new search algorithm which utilizes a *search-once-and-rank* strategy instead of IS [3], [5] and DS [6] approaches. The new approach facilitates the support for criteria that is associated with probability such as companion passengers decisions. To the best of our knowledge, our work is the first that incorporates the ridesharing request acceptance probability into the dynamic ridesharing problem.

- We develop an indexing scheme based on scheduled trips and their corresponding segments. Through incorporating interval estimates, our approach is able to respond to the queries where no historical estimates are available. Our system can also incorporate *travel time estimation* and *routes prediction*, which have been addressed elsewhere [12], [13], [14], to improve the accuracy of time and cost estimation.
- We conduct extensive experimental studies to examine the effectiveness and the efficiency of our approach and we compare it with other approaches using a real-world dataset which includes 15,784,344 trajectories of 10,357 taxis in Beijing.

The remainder of the paper is organized as follows. We define the problem and notations in Section 2. In Section 3, we formulate basic concepts and discuss the technical details. The TRIPS traffic modelling layer and the TRIPS distribution layer are discussed in Section 4. Section 5 reports the experimental results. Finally, we review the related work in Section 6 and provide some concluding remarks in Section 7.

2 THE TRIPS FRAMEWORK

In this section, we discuss the functional factors that help us design the TRIPS framework, followed by an overview of the proposed TRIPS framework.

1) Minimizing the end user's attempt to get a ridesharing: the preference of the companion passenger(s) is an important factor for the success of ridesharing. For instance, some studies suggest that behavioral parameters—such as the low rate of acceptance to the requests from male strangers by female participants who travel alone [15] and issues related with e.g., smoking [9] can affect the final decision of other riders on the same taxi. However, in this paper we do not aim to address all behavioral factors. In our model, we assume that users only decide based on their own economical benefits.

2) Maximizing the performance of the application: computationally, finding the best taxi is a complex problem. In particular, including uncertain user decisions and processing the distances of all taxis from all users are complex processes. In this paper, we particularly focus on the uncertain end users decisions and avoid duplicate rounds for performing the search when the ridesharing requests get rejected.

3) Maximizing overall savings from using the application: the savings from ridesharing is the primary goal of users when they use a the application. To maximize the savings, we improve the rate of
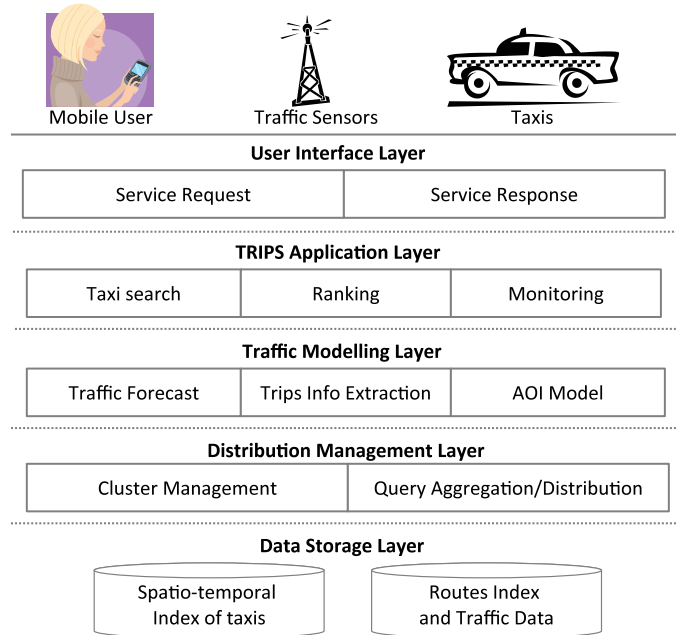

Fig. 2: TRIPS framework for taxi ridesharing service

accepted requests and the amount of savings (by reducing vehicle mileage) for the accepted queries.

TRIPS adopts a layered architecture shown in Figure 2 that consists of *User Interface* layer, *TRIPS Application* layer, *Traffic Modeling* layer, *Distribution Management* layer, and *Data Storage* layer.

There are six steps involved to find the results and return them to the user: 1) users' ridesharing queries are submitted via the *User Interface Layer*, 2) the query is passed to the *TRIPS Application Layer*, which runs for searching, ranking and monitoring the successful requests and responding user queries, 3) some of the functionalities of the previous layer such as ranking based on proximity require understanding of traffic flow and a request is sent to the *Traffic Modelling Layer*, which operates in parallel to the TRIPS application layer in order to provide the application layer with the needed traffic forecasts and identified common trips model, 4) depending on the size of the fleet that is supported by the ridesharing application, a cluster of machines can be used to manage taxi data and respond the queries, 5) once the queries are distributed in the previous step, stored data in the *Data Storage Layer*, such as the *spatio-temporal index* of taxis, the *routes index*, and the *traffic data* is retrieved and returned to the upper layer, where the results are aggregated, and 6) provided with the required data, the *Traffic Modelling Layer* is able to estimate the traffic for the duration of the trip and enable the upper layer to rank the results. Finally, ranked results are returned to the user via the *User Interface Layer*.

3 TRIPS APPLICATION LAYER

In the proposed framework, TRIPS application layer runs on top of the traffic modeling layer, facilitating modules for searching, ranking and monitoring the successful requests and responding user queries. Among the techniques mentioned above, searching the available taxis is of the greatest importance.

3.1 Taxi Search Problem Definition

Our ridesharing application continuously perceives the status of each taxi within the boundaries of the Area of Interest (AOI). Taxi status can be defined as follows:

Definition 1 (*Taxi Status*). A taxi status V represents the instantaneous state of a taxi and is comprised of a taxi ID $V.id$, a geographical location $V.l$, a list of companion passengers $V.p$, and the set of travel schedules $V.S = \{\sigma_1, \sigma_2, ...\}$ where each σ_i denotes a scheduled trip and contains an origin, destination and time windows to be at each spot. The structure of each schedule is very similar to the structure of a query. □

Ridesharing requests (queries) are generated by seeker users. We define query as follows:

Definition 2 (*Query*). A query Q is a seeker passenger's request to find a rideshare. It includes a timestamp $Q.t$ indicating when the query is submitted, a pickup point (*latitude,longitude*) $Q.o$, a delivery point (*latitude,longitude*) $Q.d$, a time window $Q.wp$ defining the time period when the passenger needs to be picked up at $Q.o$, and a time window $Q.wd$ defining the time period when the passenger needs to be dropped off at $Q.d$. The early and late bounds of a pickup window are denoted by $\underline{Q.wp}$ and $\overline{Q.wp}$. Likewise, $\underline{Q.wd}$ and $\overline{Q.wd}$ denote the bounds of the delivery window. Also $Q.u$ represents the user u who have made the query. For the sake of simplicity, each query indicates one passenger's request, but the approach can readily support multi-passengers' requests. □

Given a query Q, we would like to find the alternatives which can satisfy Q such that they can maximize the benefits for both companion and seeker users while the effort to get a rideshare is minimized. A taxi, with the corresponding status V, satisfies Q if and only if (i) $size(V.p)$ is smaller than the seat capacity of the taxi; (ii) the taxi can pick up the passenger of Q at $Q.o$ within $Q.wp$ and delivers her at $Q.d$ within $Q.wd$; (iii) the taxi can pick up and drop off the existing passengers in $V.S$ no later than the late bound of their corresponding pickup and delivery time windows.

In this paper we use a $cost$ function which depends on three main parameters: distance, time and taxi fares. The fee paid for the distance roughly is a constant number while the time variable is accountable for the extra costs that are incurred in during the waits in traffic or similar reasons. We define these two variables first and then we define the cost function. The distance between two points o and d is denoted as $\delta_{od} \in \mathrm{R}^+$, which refers to the length of the shortest path in the roads network between the two points. The travel time estimate for the same points is denoted as $\theta_{od} \in \mathrm{R}^+$, which is a measure in seconds that approximates the time required to travel from o to d. Taxi fares are positive real numbers f_c, f_d and f_t that are used to calculate the final cost. Thus, we define the cost function as follows:

Definition 3 (*Cost Function*). Cost function $cost(\delta, \theta)$ approximates the cost of a taxi based on the distance δ_{od} and the trip time estimate θ_{od} using:

$$\overline{cost(\delta_{od}, \theta_{od})} = f_c + \delta_{od}f_d + \theta_{od}f_t \qquad (1)$$
$$\underline{cost(\delta_{od}, \theta_{od})} = f_c + \delta_{od}f_d \qquad (2)$$

TABLE 1: List of important notations

Notation	Definition
AOI	The area of interest
p_i	Point $i+1$ in AOI as i start from 0.
$\delta_{p_i p_j}$	The distance between points p_i and p_j
$[\underline{i}, \overline{i}]$	Any interval value i with the minimum bound $\underline{i}$ and maximum bound $\overline{i}$
$\theta_{p_i p_j}$	The trip time estimate between points p_i and p_j at time t
$cost(\delta, \theta)$	Travel cost estimation function
Q	A query made by the seeker users
$Q.o$	The pickup point (*latitude,longitude*) of query Q
$Q.d$	The delivery point (*latitude,longitude*) of query Q
$s.o$	Pre-scheduled pickup point
$s.d$	Pre-scheduled delivery point
$Q.wp$	The pickup time window of query Q
$Q.wd$	The delivery time window of query Q
$Q.dur$	Defined as $[\underline{Q.wp}, \overline{Q.wd}]$ is the duration of a trip for Q
V	Taxi status
$V.S$	Set of scheduled trips for the given taxi V
σ	A scheduled trip
A	The set of alternatives which are displayed to the user
R^*	Maximum range of economically justifiable alternatives
idx	The segments index
u	User
$u.cost$	Ridesharing cost for user u
$u.SRcost$	Sole ride cost for user u
$\mathcal{V}$	Any set of taxi trajectories
$P(u', Q)$	The probability of user u' accepting a ridesharing offer based on Q

Where o and d denote the origin and destination of a trip, f_c is a constant pickup fee, f_d denotes the constant distance rate and f_t denotes traffic and waiting rate. □

We also consider the decisions of the companion users to minimize the effort for users by reducing the number of attempts needed to get a ridesharing. We suppose that every user makes decision based on their own savings from the ridesharing. Thus, for a new passenger u and every companion passenger u', the following statement should be true:

$$\begin{aligned} u.cost &< u.SRcost; and \\ u'.cost &< u'.SRcost \end{aligned} \qquad (3)$$

Table 1 summarizes the most important notations that we use in the paper. Other notations will be covered in the rest of the paper.

3.2 Economic Search Margin

An important question in taxi search is that how far we can go to look for the available taxis. In the incremental search, the area is extended as far as possible until the taxis that satisfy the request are found. To reduce the search area, decremental search uses the area that includes all of the economically justifiable taxis as the maximum search area. The margin of this area is called *economic search margin*. Before discussing the economic search margin in Section
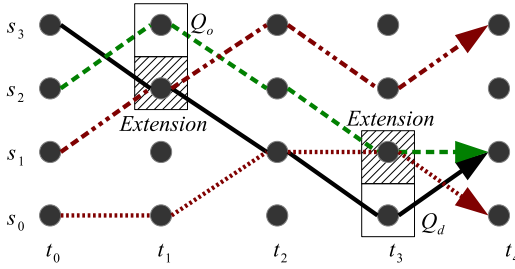
Fig. 3: Search area extension for query Q

3.2.2, we first introduce the travel sequences that are used in measuring the economic benefits in Section 3.2.1.

3.2.1 Travel Sequences

The travel sequence of a taxi is the transition of locations it stops when picking up and delivering pre-arranged passengers and new seeker users. However, there are two issues of directly using the exact locations of pickup and destination points of the passengers to construct the travel sequences: i) it is not assured that there are taxis in each exact location; ii) the passengers that already book the taxi might not want to share the ride with others. To tackle these issues, we seek alternative taxis to meet the request of both the passengers and the ride-sharers by extending the search areas.

Figure 3 illustrates an example of area extension. The four sequences s_0 to s_3 specify the transition of four taxis (with different lines) on four points (the black nodes) during the time window t_0 to t_4 before a new query Q is given. Q requests Q_o (i.e., the fourth point from bottom) in t_1 for the pickup and Q_d (i.e., the first point from bottom) in t_3 for the drop off. As shown in the figure, in this scenario no taxi in the pickup point (specified with s_2) is found in the drop off point, while the taxi in the destination (specified with s_3) is not in the origin if the exact points in the query are used. As a result, no taxi can be found for Q. In this situation, to find a compromise solution, we extend the search area to find the nearest taxi that satisfies the query. After applying two extensions (i.e., the journeys starting from neighboring points next to Q_o and ending in point next to Q_d respectively) for Q, two taxis can found alternatively. Then one of the taxis will be selected based on the extra distance required by taxis to get to the points $Q.o$ and $Q.d$ [3]. As each ridesharing request needs to be evaluated by the companion passengers and can possibly be rejected finally, we need to further expand the area to search for available taxis.

After obtaining all the alternatives, we can rank and sort them based on economical merits to benefit users with less effort. In order to establish a search and rank strategy, we analyse the possible sequences of the schedules and measure their economical cost.

Let $Q.o$ and $Q.d$ respectively denote the origin and the destination of a query, $p.o$ and $p.d$ denote a scheduled trip's origin and destination, $s.o$ and $s.d$ denote a pre-scheduled trip's origin and destination,[1] and $V.l$ is the current location

---

1. For the sake of simplicity, we only consider one travel schedule for existing passengers. But the approach can be generalized to support sequences with multiple pickup/drops.
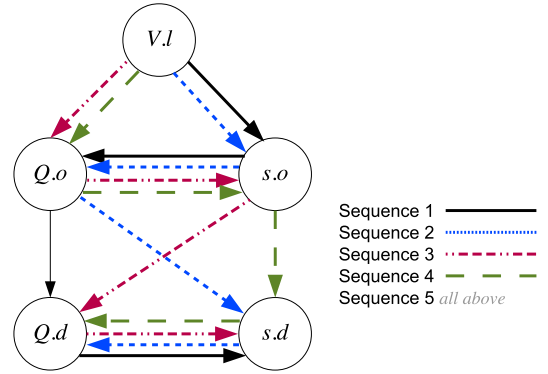

Fig. 4: Possible schedule sequences for a query

of the taxi. There are six possible travel sequences for the ridesharing request Q as shown in Figure 4:

- Sequence 1: $\{V.l, s.o, Q.o, Q.d, s.d\}$
- Sequence 2: $\{V.l, s.o, Q.o, s.d, Q.d\}$
- Sequence 3: $\{V.l, Q.o, s.o, Q.d, s.d\}$
- Sequence 4: $\{V.l, Q.o, s.o, s.d, Q.d\}$
- Sequence 5: if one or more time windows overlap, any order for pick up/delivery is possible
- Sequence 6: if the companion traveller is already on the taxi

The cost of these travel sequences are formulated as follows:

Sequence 1: Respectively, pre-scheduled pickup, new user pickup, new user delivery, pre-scheduled delivery. Assuming that the user shares the ride with m other passengers, the riding cost can be formulated as follows (please refer the notations in Table 6):

$$u.cost = \frac{cost(\delta_{od}, \theta_{od})}{(m+1)} \qquad (4)$$

$$u'.cost = \frac{cost(\delta_{io}, \theta_{io})}{m} + \frac{cost(\delta_{od}, \theta_{od})}{m+1} + \frac{cost(\delta_{dj}, \theta_{dj})}{m} \qquad (5)$$

Sequence 2: Respectively, pre-scheduled pickup, new user pickup, pre-scheduled delivery, new user delivery. The riding cost can be formulated as follows:

$$u.cost = \frac{cost(\delta_{oj}, \theta_{oj})}{m+1} + cost(\delta_{jd}, \theta_{jd}) \qquad (6)$$

$$u'.cost = \frac{cost(\delta_{io}, \theta_{io})}{m} + \frac{cost(\delta_{oj}, \theta_{oj})}{m+1} \qquad (7)$$

Sequence 3: Respectively, new user pickup, pre-scheduled pickup, new user delivery, pre-scheduled delivery. The riding cost can be formulated as follows:

$$u.cost = cost(\delta_{oi}, \theta_{oi}) + \frac{cost(\delta_{id}, \theta_{id})}{m+1} \qquad (8)$$

$$u'.cost = \frac{cost(\delta_{id}, \theta_{id})}{m+1} + \frac{cost(\delta_{dj}, \theta_{dj})}{m} \qquad (9)$$

Sequence 4: Respectively, new user pickup, pre-scheduled pickup, pre-scheduled delivery, new user delivery. The riding cost can be formulated as follows:

$$u.cost = cost(\delta_{oi}, \theta_{oi}) + \frac{cost(\delta_{ij}, \theta_{ij})}{m+1} + cost(\delta_{jd}, \theta_{jd}) \quad (10)$$

$$u'.cost = \frac{cost(\delta_{ij}, \theta_{ij})}{m+1} \quad (11)$$

Sequence 5: One or more overlapping schedules. In this case, we cannot find a deterministic order for pickups and deliveries as the taxi driver may pick any order in advance. Thus, as the level of uncertainty is higher in this case, the estimated cost of each ride can be formulated as follows:

$$\underline{u.cost} = \frac{cost(\delta_{od}, \theta_{od})}{(m+1)} \quad (12)$$

$$\overline{u.cost} = \overline{cost}(\delta_{oi}, \theta_{oi}) + \frac{\overline{cost}(\delta_{ij}, \theta_{ij})}{m+1} + \overline{cost}(\delta_{jd}, \theta_{jd}) \quad (13)$$

$$\underline{u'.cost} = \frac{cost(\delta_{ij}, \theta_{ij})}{m+1} \quad (14)$$

$$\overline{u'.cost} = \frac{\overline{cost}(\delta_{io}, \theta_{io})}{m} + \frac{\overline{cost}(\delta_{od}, \theta_{od})}{m+1} + \frac{\overline{cost}(\delta_{dj}, \theta_{dj})}{m} \quad (15)$$

Sequence 6: If the companion passenger has already started his/her trip and is already on the taxi then the taxi will not get to the $p.o$ and the sequence starts with $V.l$. In this case, depending on the order of the destinations of the schedules, the estimated cost of each ride can be formulated similar to the Sequences 1 and 2. However, in both cases the $V.l$ will replace $p.o$.

3.2.2 Radius of the Economic Search Margin (R^*)

One of the problems that is associated with the incremental search approach is that the search area is gradually extended until the nearest taxi with enough space is found. Thus, it is possible that for a number of searches, we get results which should travel a long distance to reach the seeker user making it is not efficient for the companion passengers. We use the following theorem to set up this concept.

Theorem 3.1. *Given the Query Q, where the user intends to travel from o to d, a taxi that can pick the user iff it does not have any other trip with the origin and destination further than the following distance from the points o and d:*

$$R^* = \delta_{od} \quad (16)$$

where δ_{od} denotes the distance that the user wants to travel and R^* is the economically justifiable radius. A taxi with scheduled trips further this distance is not within the economically justifiable range.

Proof. We can prove the above theorem by using costs and benefits analysis for all possible schedule sequences. Based on the mutual benefit principle, by summing up the total cost for all users (e.g., all passengers), we have:

$$u.cost + m * u'.cost < u.SRcost + m * u'.SRcost \quad (17)$$

This statement holds for all of the possible schedule sequences. For instance, for Sequence 1, we have:

$$u.cost + m * u'.cost = cost(\delta_{oi}, \theta_{oi}) + cost(\delta_{ij}, \theta_{ij}) + cost(\delta_{jd}, \theta_{jd}) \quad (18)$$

If we make a false assumption that $R^* > \delta_{od}$, we will have the following:

$$u.cost + m * u'.cost \geq SRcost \quad (19)$$

where $SRcost$ denotes $u.SRcost + m * u'.SRcost$. Equation (19) in fact contradicts the mutual benefit principle and cannot be correct. Theorem 3.1 is therefore proved. □

3.3 Proposed Taxi Search and Ranking

We introduced the proposed taxi searching and ranking methods in this section.

3.3.1 Taxi Search

Upon receiving a query from a user, the taxi search module returns all possible taxis which can satisfy the query by considering uncertainty. Unlike the incremental and decremental search approaches which increases or decreases the search area until a satisfied taxi is found, in TRIPS, we develop a new strategy that fixes the search area and then loops all the travel schedules. This approach performs in a *search once to get all* manner and we name it Fixed Search (FS). To improve the FS, we further develop a Index Powered Fixed Search (IPFS) that leverage index to facilitate the search and introduce the concept of segment.

Fixed Search (FS). Algorithm 1 introduces the fixed search procedure. First, R^* is initialized to prevent searching unnecessary locations (line 1). Then the algorithm loops through every scheduled trip of each taxi (lines 2-3) and checks if the duration of trips overlap at pickup point, the taxi is added to T_1. Also, if they overlap at the destination, the taxi is added to the T_2 set (lines 4-7). Later, common taxis in T_1 and T_2 are added to the set of candidates (lines 8-10). As only one taxi is finally chosen, a result set that consists numerous candidate taxis is not suitable result for the users. These taxis need to be ranked and sorted based on the gain and the possible acceptance rate by other users. Thus, we call the RANK function (discussed later) and return the best taxi as a result (lines 11-12).

The complexity of the fixed search depends on two factors: the number of taxis ($|taxis|$) and the number of scheduled trips (n). Thus the order of fixed search approach is $\mathbf{O}(|taxis|.n)$. As a result, with the growth of the number of taxi schedules, the fixed search will become inefficient as it is shown by our experiments (Section 5).

Index Powered Fixed Search (IPFS). Before presenting the IPFS algorithm, we first set up the indexing scheme. To create the index, similar to T-Share [3], we divide the spatial domain of moving taxis, which we refer to as the Area of Interest (AOI), into a finite set of rectangular cells. Each cell is called a segment and is denoted by seg. The number of segments in an AOI is often limited and constant. Also, the

---

Algorithm 1 FIXED_SEARCH

Require: Q user's query
Ensure: T set of available taxis
1: Let $R^* \leftarrow \delta_{od}$ Setting the R^* to avoid searching the unnecessary area
2: **for all** $taxi \in taxis$ **do**
3:    **for all** $\sigma \in taxi.V.S$ **do**
4:       **if** $i = \sigma.pickup$ and $\delta_{io} \leq R^*$ **and** $Q.dur$ overlaps($\sigma.dur$) **then**
5:          Add $taxi$ to T_1
6:       **if** $j = \sigma.destination$ and $\delta_{jd} \leq R^*$ **and** $Q.dur$ overlaps($\sigma.dur$) **then**
7:          Add $taxi$ to T_2
8: **for all** $taxi \in T_1$ **do**
9:    **if** $taxi \in T_2$ **then**
10:       Add $taxi$ to T
11: $T \leftarrow$ RANK (T)
12: **return** $T.first$

Algorithm 2 INDEX_POWERED_FIXED_SEARCH

Require: Q user's query, AOI list of segments in AOI
Ensure: T set of available alternatives
1: Let $R^* \leftarrow \delta_{od}$ Initializing the R^* to avoid searching the unnecessary area
2: **for all** segment $seg \in AOI$ **do**
3:    **for all** $trip \in seg.idx$ **do**
4:       **if** $\delta_{seg,o} \leq R^*$ **and** $Q.dur$ overlaps($trip.dur$) **then**
5:          Add $idx.taxi$ to T_1
6:       **if** $\delta_{seg,d} \leq R^*$ **and** $Q.dur$ overlaps($trip.dur$) **then**
7:          Add $idx.taxi$ to T_2
8: **for all** $taxi \in T_1$ **do**
9:    **if** $taxi \in T_2$ **then**
10:       Add $taxi$ to T
11: $T \leftarrow$ RANK (T)
12: **return** $T.first$

Algorithm 3 RANK

Require: Q user's query, T the set of taxis
Ensure: A_r ranked list of alternatives
1: **for all** $taxi \in T$ **do**
2:    **for all** $\sigma \in taxi.V.S$ **do**
3:       Let seq be the sequence of $Q.wp$ and $Q.wd$ comparing with $\sigma.wp$ and $\sigma.wd$
4:       Calculate $Q.u.SRcost$, $Q.u.cost$, $Q.u.saving$ based on seq
5:       Calculate $\sigma.u.SRcost$, $\sigma.u.cost$, $\sigma.u.saving$ based on seq
6:       Update $P(\sigma.u, Q)$ based on seq
7:       Update $Q.u.totalSRcost$, $Q.u.totalCost$ and $Q.u.totalSaving$
8:       Update $\sigma.SRcost$, $\sigma.cost$ and $\sigma.saving$
9:    Set the saving and possibility to $taxi$
10: Update global tree of possibilities T by BUILD-TREE($Q,A,nil,nil,0$)
11: Let $A_r \leftarrow$ SORT (A, T) be the set of sorted alternatives
12: **return** A_r

---

number of scheduled trips which begin from or end to a segment are limited. Therefore, we introduce a segment based index denoted by $seg.idx$, which keeps the set of schedules relevant (pick up or delivery) to seg. The index contains two sets of entries. The first set is the scheduled trips from the segment and the second contains the scheduled trips to the segment. The records in the index are updated whenever a new trip is scheduled and removed whenever a trip is finished. Thus, the index is updated during schedule set up and schedule removal steps without notable extra workload on the system. The complexity order of the index update algorithm, if implemented separately, is $O(\sum |V.S|)$.

The IPFS approach is presented in Algorithm 2. Similar to the fixed search, IPFS starts with initializing R^* (line 1). The algorithm loops through the index entries for each segment within the range of $R*$ instead of the schedules of each taxi (lines 2-4). Then for each segment within the *economical search margin* (i.e., the area that includes all of the economically justifiable taxis), if a scheduled trip is found in the segment index entry, the corresponding taxi will be added to the set of found taxis at origin denoted by T_1 and/or destination denoted by T_2 (lines 5-8). Only taxis appearing in both sets (T_1 and T_2) are included in the output (T), ranked similar to the FS algorithm and finally returned (lines 9-13).

The complexity of the IPFS algorithm depends on two factors: the number of index entries (n) and the number of segments ($|AOI|$). Thus the order of FS approach however depends on the number of taxis as well. However, the execution cost of this algorithm will be lower than the FS algorithm due to the fact that only a small portion of the scheduled trips are processed in each round. With the traditional approaches, we need to repeat the search multiple times to find the optimal solution and thus, I/O access rate increases with the number of taxis and the number of cells [3]. However, in IPFS I/O access rate for searching taxis remains at one.

3.3.2 Ranking

After all the available taxis are searched, we rank the them and return the best one to the users. Ranking can be challenging when it comes to uncertain data. The score for each available taxi will be represented as an interval. Hence,

to rank and sort them, different possible orders must be considered. We propose the RANK algorithm in Algorithm 3 which works in the following order: 1) the algorithm starts by calculating the score for each alternative. For each active scheduled trip (line 2) of each available taxi (line 1), we determine the sequence (seq) of the new user's trip and the previously scheduled trip (line 3) and update users sole ride cost $u.SRcost$, shared ride cost $u.cost$, the amount of saving $u.saving$ and the possibility of accepting the ridesharing request $P(u, Q)$ based on the savings in lines 4-9; 2) In the next step (line 10), we pass the query Q and the set of scored available taxis A to BUILD-TREE algorithm (Algorithm 4) which updates the global tree of possible worlds; 3) Then, the alternatives will be sorted based on the tree of possible worlds and the result will be returned (lines 11 and 12).

The complexity of RANK algorithm in the worst case is $\mathbf{O}(|taxis|.n)$ if $|taxis|$ and n represent the number of the taxis and the number of scheduled trips. However, due to the fact that usually only a small subset of the taxis set is received from the search algorithm, the runtime and order of RANK algorithm are negligible. Moreover, using segment schedules' index can also improve this algorithm by reducing the number of accessed schedules.

The BUILD-TREE algorithm is designed using the *probabilistic partial order* defined as follows:

Definition 4 (*Probabilistic Partial Order [16]*). Let $A = \{a_1, ..., a_n\}$ be the set of the available taxis with their scores, and O be the set of orders of alternatives. The probabilistic partial order $PPO(A, O)$ is a set with $(a_i, a_j) \in O$ iff a_i precedes a_j. □

Algorithm BUILD-TREE recursively builds the tree of possible worlds, which is a globally defined variable, and can stop at the specified level, if any (line 1). Building each level starts by finding sources (lines 2-4), which are available taxis that dominate others by their scores, and ends up with passing the generated probabilistic partial order to the next level (line 5). The complexity of this algorithm only depends on the number of available taxis received from the RANK algorithm, and as there are usually a small number of taxis, we expect that the complexity and runtime of the BUILD-TREE algorithm would be negligible.

4 TRAFFIC MODELLING AND DISTRIBUTION MANAGEMENT

The traffic modelling layer operates in parallel to the TRIPS application layer (details in Section 3) in order to provide the application layer with the needed traffic forecasts and identified common trips model. The *AOI model, the* trips information extraction, and the *traffic forecast* are the three main modules of this layer.

AOI Model: The underlying roads network can be modeled via different approaches such as *R-tree* and partitioning using a grid network. In our work, a grid partitioning system similar to [3] is developed in order to avoid high cost indexing level. However, the indexing system can easily be upgraded if needed.

Trips Information Extraction: Trips information extraction module is designed to obtain the regular trips that taxis undertake within different locations in a certain period of time. For this purpose, we use the same algorithm as in [6] which provides the actual upper and lower bounds from the historical data. In our study we use these results to avoid dealing with technical complexity over the accuracy of predicted travel times. However, in application it can simply be replaced by prediction algorithms such as [14].

Traffic Forecast: The traffic forecast module can exploit several techniques to forecast traffic because many approaches have been proposed for this task in the literature [17], [18]. Although the traffic forecast model is not the focus of this study, we briefly discuss in the following on how traffic index is generated in our work.

To generate a traffic index, the speed of taxis moving along a specified trip is taken as an index. The traffic load in a route between two points can have direct relationship with the speed of cars passing over that route if they are moving.

Algorithm 4 BUILD-TREE

Require: $PPO(A, O)$, Treenode n, *level*
1: **if** $level \leq max - level$ **then**
2:    **for all** sources $taxi \in A$ **do**
3:       Add $taxi$ as a new *child* to children
4:       Let $PPO \leftarrow PPO(A, O)$ after removing $taxi$
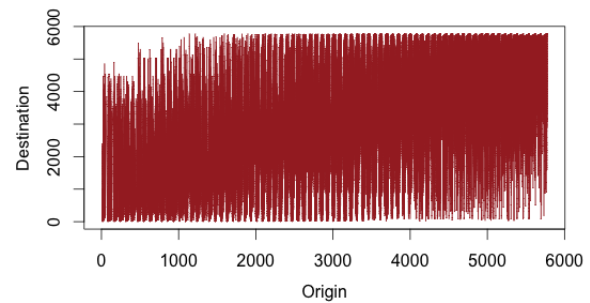5:   BUILD-TREE($Q,A,PPO,child,level + 1$)

Fig. 5: Availability of the index for all origins and destinations

A query over historical data record can easily provide the margins of each entry in traffic index and trip time matrices. Thus $\overline{\theta_{p_i p_j}} = max\{\theta_{p_i, p_j}\}$, $\underline{\theta_{p_i, p_j}} = min\{\theta_{p_i, p_j}\}$.

The distribution management layer handles two main tasks: the *cluster management* and the *query distribution/aggregation*.

The cluster management module maintains taxis in a set of object clusters. The purpose of clustering the taxis is to reduce the searching space of the available taxis, as both of the two search algorithms we proposed in Section 3.3.1 requires the examination of the schedules of all the taxis in the searching space. Formally, an object cluster is defined as:

Definition 5 (*Object Cluster*). Object cluster is a set of tables containing the data of moving objects (taxis) $\{taxis_i | i \in D(taxis)\}$, where the member items share one or more common values in their status V such as their location or corresponding ID. The set of clusters can be defined as: $C = \{c_i | i = 1, 2, ..., X\}$, where X is the number of clusters and each cluster can be shown as $c_i = \{o_j | j = 1, 2, ..., Y\}$, where Y is the size of the cluster. □

The query distribution and aggregation module is responsible for dividing an incoming query, denoted as $Q = \{q_i | i = 1, 2, ..., X\}$ in which each q_i is submitted to the corresponding cluster c_i for further query processing (i.e., taxi search). Later, the returned results are aggregated to compose the final result set.

5 EVALUATIONS OF TRIPS

We implement the proposed TRIPS framework and conduct extensive experiments using a real world dataset to study its effectiveness and efficiency.

5.1 The Dataset

We evaluate the performance of TRIPS with both the real world data and the simulated data. The datasets are described in this section.

5.1.1 Real World Data

The real world dataset we used contains the records of raw GPS trajectories for 10,357 taxis in the city of Beijing, China over 3 months. The average sampling interval of the data set is 3.1 minutes per point and the average distance between two consecutive points is about 600 meters. Real user trajectories were gathered on 30 drivers on a 2-month
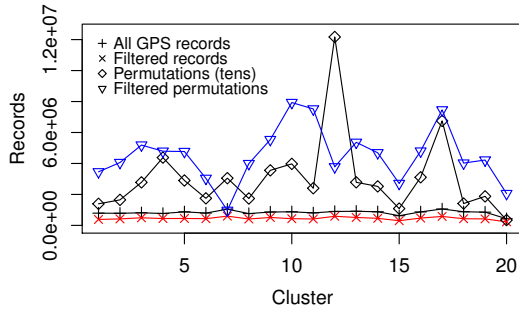
Fig. 6: Dataset size in each step during data preparation

driving history. The average sampling interval is about 10s [12], [19].

In addition to raw trajectories data, we need the details of previous trips in order to generate the indexes and prepare traffic forecast data. Few examples of the records in this dataset are as follows:

1,2008-02-02 15:36:08,116.51172,39.92123
1,2008-02-02 15:46:08,116.51135,39.93883
1,2008-02-02 15:46:08,116.51135,39.93883

To prepare the data for our experiments, we apply a number of data refinement steps which are described in the following. To create the index, we first divide the AOI into a 76×76 segments grid and map each GPS reading to its corresponding segment. Next, we remove the records that are located outside of the AOI. Then, for every record we use the distance and the time elapsed from the last record to calculate the minimum speed of a taxi. Since we do not have the details of previous trips for the taxis, for every taxi trajectory we calculate the permutations of successor records. Each permutation can be considered as a possible trip unless it represents a trip to the initial origin point. To have a set of permutations with a reasonable size, we dismiss the records with speeds less than 1 km/h (i.e., taxi stops or traffic jams). Figure 5 shows the index availability for all origins and destinations in the AOI. Each dark point indicates that the time estimation for the corresponding segments pair exists. The density in the central areas is higher than the density of indexed data in the surrounding areas while some of the segment pairs have no data. The purpose of this figure is to demonstrate that our defined index can practically cover the majority of the segment pair of the map.

5.1.2   Simulation Data

We also generate a synthetic dataset for some of the evaluations. The dataset is set to be similar to the real world dataset that generates a 76×76 grid of segments. It is generated using key parts of a random taxi query generator [3]. We also use 10,000 taxis with limited capacity (up to 4 people) to simulate a realistic scenario.

5.2   Search Approaches for Comparison

To find the taxis with the least extra distance, area extension is often applied to the origin and destination areas. The reason is that for most of the queries, no taxi is found to exactly match the specified requirements. This approach is

regarded as incremental search approach. Instead of increasing the search area, we can decrease the search area in each step by using the economic search margin. This leads to the decremental search approach. We introduce these two types of taxi search for comparison to our proposed methods.

5.2.1   Incremental Search (IS)

Incremental search has been used in designing dynamic ridesharing applications such as T-Share [3]. Algorithm 5 shows a generic incremental search procedure for taxis based on the algorithm in T-Share. The input of the algorithm is the given query Q and the extended areas of origin and destination, which in the first round would be equal to $Q.o$ and $Q.d$. First, the taxis at extended origin area and extended destination area are queried and stored in T_1 and T_2 respectively (lines 1-2). The GET_TAXIS function loops through all taxis to find the right ones. The list of the common taxis in the two sets are stored in the T set (lines 3-5). Then, if T is empty (lines 6-9), the area is expanded and search is recursively called with the expanded origin and/or destination areas and the same query. Otherwise, if a common taxi is found in the same area, it is returned (lines 10-11). The complexity of this algorithm depends mainly on three factors: the number of segments ($|AOI|$) in AOI, the number of taxis ($|taxis|$), and the number of scheduled trips n. Thus the overall complexity order of the incremental search algorithm is $\mathbf{O}(n.|AOI|.|taxis|^2)$.

Although incremental search has been shown its effectiveness in [3], it has one main drawback that it does not support the search margin as it continues the area extension operation until the first taxi is found. However, when no taxi can be found, it will expand to the whole available area, rendering tremendous computation cost. Thus, decremental search are proposed to fill this gap [6].

5.2.2   Decremental Search (DS)

The decremental search approach is proposed to decrease the cost of the search procedure [6]. To initialize, the decremental search requires the total extended area that includes all of the economically justifiable taxis. We call it *Economic Search Margin* and denote its radius as R^*.

The details of this approach are shown in Algorithm 6. The inputs to the algorithm are the given query Q and the extended areas of origin and destination. In the first step of this algorithm, we check if each of the origin and

---

Algorithm 5 INCREMENTAL_SEARCH

Require: $Q, origin, dest$
Ensure: T set of taxis which satisfy the query
1: Let $T_1 \leftarrow$ GET_TAXIS ($origin$,$Q.wp$)
2: Let $T_2 \leftarrow$ GET_TAXIS ($dest$,$Q.wd$)
3: **for all** $taxi \in T_1$ **do**
4:     **if** $taxi \in T_2$ **then**
5:        Let $T \leftarrow T \cup taxi$ add taxi to output list.
6: **if** T is empty **then**
7:     Let $origin \leftarrow$ EXPAND ($origin$,$Q.o$) extending the search area.
8:     Let $dest \leftarrow$ EXPAND ($dest$,$Q.d$) extending the search area.
9:     **return** INCREMENTAL_SEARCH (Q,$origin$,$dest$)
10: **else**
11:     **return** T

(a) Average query response time   (b) Average search time   (c) Average rank time
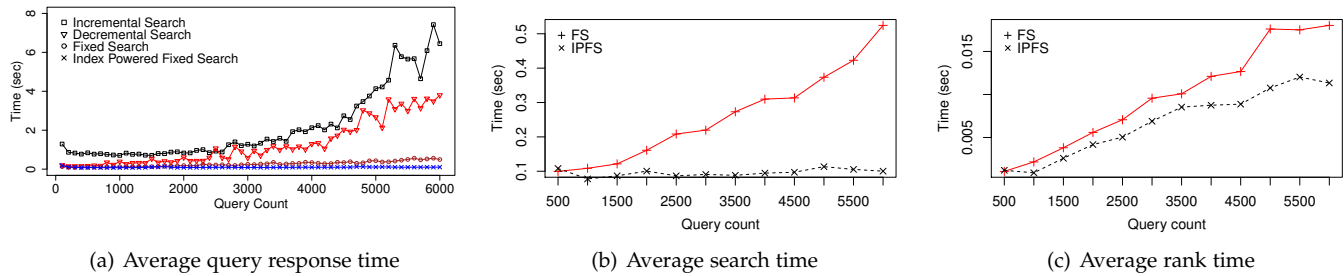
Fig. 7: Time comparison for different approaches.

destination areas are limited to only their initial values or in other words, none of them is an extended area. In this case, we cannot split them further and the set of common taxis in the specified time windows is returned (lines 1-2). The GET_TAXIS function is similar to the same function in the incremental approach. If the search areas are already extended, the rest of the process is applied as follows. We search the areas with the given time frames for the taxis and store them in T_1 and T_2 (lines 3-4). The set of common taxis which appear at both T_1 and T_2 are stored in T (lines 5-7). Next, if T is not empty, which means that some taxis can be found, we shrink the search areas and recursively call the decremental search with the new parameters and then store them in T' (lines 8-11). If no common taxi is found, the taxi search algorithm can stop the further recursion. The results of recursion will replace current set of T only if they are not empty (lines 12-13) and finally T is returned.

Due to the early stop feature, decremental search can potentially increase the speed of the search process by stopping further recursion when no suitable taxi is found in the economically justifiable range. As a result, the decremental approach can increase the efficiency of incremental search approach.

5.3 Evaluation Results

5.3.1 Efficiency of TRIPS

We analyse two measurements for the performance and the scalability of TRIPS: i) search space, and ii) its ability to respond to a heavy workload.

Algorithm 6 DECREMENTAL_SEARCH

Require: $Q, origin, dest$
Ensure: T set of taxis which satisfy the query
1: **if** $origin$ equals $Q.o$ **and** $dest$ equals $Q.d$ **then**
2:      **return** $T \leftarrow$ GET_TAXIS $(origin, dest, Q.wp, Q.wd)$
3: Let $T_1 \leftarrow$ GET_TAXIS $(origin, Q.wp)$
4: Let $T_2 \leftarrow$ GET_TAXIS $(dest, Q.wd)$
5: **for all** $taxi \in T_1$ **do**
6:      **if** $taxi \in T_2$ **then**
7:          Let $T \leftarrow T \cup taxi$ add taxi to output list.
8: **if** T is **not** empty **then**
9:      Let $origin \leftarrow$ SHRINK $(origin, Q.o)$ shrinking the search area.
10:     Let $dest \leftarrow$ SHRINK $(dest, Q.d)$ shrinking the search area.
11:     Let $T' \leftarrow$ DECREMENTAL_SEARCH $(Q, origin, dest)$
12: **if** T' is **not** empty **then**
13:     Let $T \leftarrow T'$
14: **return** T

Record Size in Data Preparation. For the first measurement, we use the real world dataset. The size of search space can greatly affect the performance of a search procedure. One of the advantages of using segments index is that it fixes the maximum size of search space. For any number of given trips data in the input, the size of trip information index is equal to or less than $|AOI|^2$. Figure 6 shows the size of the dataset in each step of the dataset preparation. The number of filtered records is less than the original raw records as we prune the records out of AOI. The number of records after the permutation calculation increases drastically, thus to show clearly, we divided the number by ten. After the filtering, the number of records is reduced and is slightly higher than the number of records before filtering. We measure the records size when varying the number of clusters and observe that after permutation calculation, the cluster number will affect the number of records. But we cannot identify specific patterns from the result, indicating the record size is insensitive to the cluster number.

Time Consumption Comparison. To examine TRIPS's ability to respond to a heavy workload, we first examine the time used when the number of queries increases. In this experiment, we use the synthetic dataset and sampled 6,000 queries which are processed by the system. The baseline approach is an improved version of T-Share which uses the incremental search. The improved version applies extension to the search area by adding all the surrounding segments (instead of one by one segment strategy used by e.g., T-Share) in each iteration. Figure 7 depicts the time consumption comparison for the baseline work and TRIPS with different search approaches. Figure 7(a) examines the total query processing time among incremental search (IS), decremental search (DS), fixed search (FS) and index powered fixed search (IPFS). As shown in the figure, the IPFS algorithm is the most efficient one, consuming the least average query response time. IS in general is the most time-consuming one. When the number of query increases, the time used by IS and DS increases accordingly because these two search algorithms adjust the search area for each of the query one by one. FS and IPFS have negligible increase in time usage as they have fixed search area and can record the result for many queries at one time. We also measure the amount of time spent on *search* and *rank* stages separately and for each of them, we compare the runtime between using and not using the segments index. Figure 7(b) compares the average search time between FS and IPFS for the same number of queries. As it shows, while the IPFS takes less

than 0.1 second until the end of the experiment while the FS search time gradually increases and reaches to 0.5 seconds. This is mainly due to the constant size of the search space in the IPFS algorithm. The RANK function can benefit from the segments index as well. Figure 7(c) compares the average rank time for FS and IPFS. The use of segments index reduces the increment in average ranking time by nearly 50%. The average rank time for index powered approach is nearly 10 milliseconds per query where without using the index, it can take up nearly 20 milliseconds for the last set of queries.

Average Number of Alternatives. To further examine the TRIPS's ability to respond to a heavy workload, we conduct one more experiment to analyse the *average number of alternatives*. Figure 8 depicts the results. As shown in the figure, the number of alternatives (i.e., available taxis) for FS and IPFS increase when the number of queries increases, while for IS and DS, the number remains the same. The reason is that IS and DS will stop when one available taxi is found. But FS and IPFS consider all the available taxis within the economic search margin. The latter two approaches will be in overall more efficient as they can answer multiple queries for one scan.

5.3.2 Effectiveness of TRIPS

In this experiment, we investigate the effect of considering uncertainty in the total savings of the ridesharing process. The riders who have booked the taxi service earlier can accept or reject the incoming ridesharing requests. The decision is made based on the ratio of their own savings from the ridesharing to their initial cost:

$$P(u', Q) \propto \frac{u'.SRcost - u'.cost}{u'.SRcost} \qquad (20)$$

where u' is a user who has previously booked the taxi for a time which overlaps with the pickup and delivery time window of Q, $P(u', Q)$ is the possibility of u' accepting a new ridesharing quest based on Q, $u'.cost$ is the cost after accepting the ridesharing and $u'.SRcost$ denotes the sole ride cost for u' before accepting the rideshare. Higher $P(u', Q)$ indicates higher acceptance. We choose the one with the highest $P(u', Q)$ to accept.

To show the effectiveness of TRIPS, we compare the two methods that with and without considering the acceptance probability in the rank step. Note that we use IPFS as the search approach in this evaluation. Figure 9 depicts the comparison results. Nearly 1,000 ridesharing requests were

Fig. 8: Number of processed alternatives

successful in finding a set of suitable alternatives. Figure 9(a) shows that considering the companion passenger decision factor increases the acceptance rate of the companion users.

To assess the amount of financial benefits for users, we use the following formula to calculate the taxi fares in Chinese Yuan (CNY)[2]:

$$\overline{cost(\delta, \theta)} = 12 + \delta * (2.2) + (\overline{\theta} - \underline{\theta}) * (0.01)$$
$$\underline{cost(\delta, \theta)} = 12 + \delta * (2.2) \qquad (21)$$

Figure 9(b) shows the upper and lower bounds for the estimated cumulative savings. The possible amount of savings increases due to the increment in acceptance rate. As the amount of uncertainty increases throughout the time, the upper and lower bounds of the two estimations may interfere. But from the experimental results, we can see that the total savings remarkably increase by using our TRIPS framework. This is due to the reason that our approach considers the most probable and most efficient alternatives rather than the most efficient ones only.

6 RELATED WORK

In this section, we overview the research activities that are related to the research work presented in this paper. We firstly reviewed the approaches that are closest to our work. Then we review other related literatures.

T-Share [3], [5] is the main work comparable with our TRPIS. T-Share is a mobile-cloud based real-time taxi-sharing system. It is designed for drivers to accept taxi rider's requests based on proper schedules. The taxi search part is similar to incremental search. However, with returning the whole set of available taxis, ranking and sorting taxis will become complex and more time consuming as it depends on the number of taxis. Sharek [20] is another candidate which can be compared to our search approach. The main purpose of Sharek is to design a scalable dynamic ridesharing system for dynamic ridesharing which allows riders requesting the ridesharing service to indicate the maximum price they are willing to pay and the maximum waiting time before being picked up. However, it does not take the response time and other scheduled trips into account. This approach also uses an incremental search without indexing. Thus, this approach can be less efficient than the fixed search and index powered fixed search approaches in TRIPS.

6.1 Dynamic Ridesharing

Ridesharing has been actively studied in past few years. Dynamic ridesharing is generally described as an automated system that facilitates drivers and riders to share one time trips close to their departure times/places and can be characterized with features like dynamic, independent, cost-sharing, prearranged and automated matching [2].

Most of the proposed applications have not been examined for either real-life and/or large-scale datasets. These solutions are developed based on a variety of techniques and approaches such as auction negotiation [21], analyzing social connections in SRSS [22], multi source-destination path planning [23] and even cloud computing [24]. and [8]
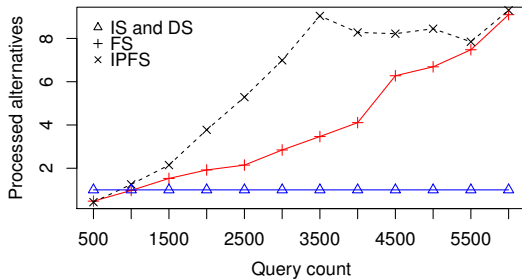
2. http://www.numbeo.com

(a) Number of the accepted ridesharing requests
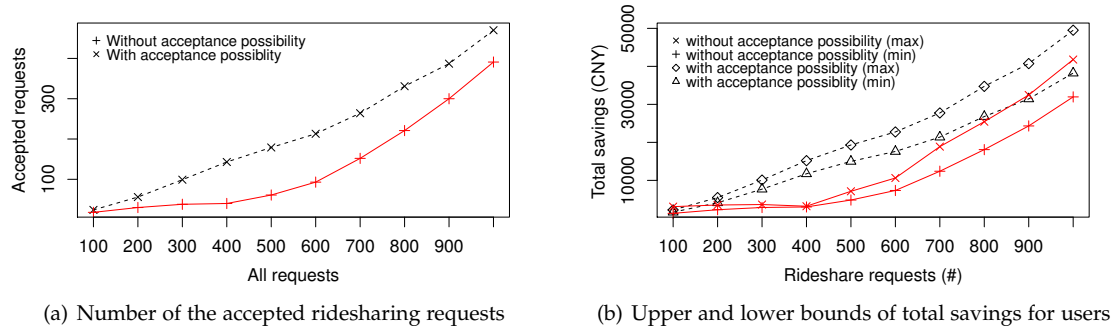

(b) Upper and lower bounds of total savings for users

Fig. 9: Effectiveness evaluation of TRIPS

TABLE 2: Taxi speed estimation based on GPS reading analysis

All	Stopped $v < 1\ km/h$	High Probability $1 \leq v < 90\ km/h$	Low Probability $90 \leq v < 200\ km/h$	Impossible $v \geq 200\ km/h$
15,784,344	6,419,947	9,280,967	32,283	51,147

that focuses on different criteria. Some other very recent solutions such as T-Share [3], [5], Noah [25] and kinetic tree algorithm [26] treat dynamic taxi ridesharing as finding k-nearest neighbours (kNN) problem although it does not cover the whole challenges and minimizing the system wide travel time and travel distance. However, although using this approach can enormously reduce the complexity of the problem, using kNN for dynamic ridesharing is not a suitable approach as k is unclear. STaRS [27] focuses on reducing the computation complexity. It leverages cache-coherent shortest path index and parallelism to achieve the efficient taix ride-sharing analysis. CallCab [28] deploys a generic Map-Reduce measure to tackle the raw dataset of 14,000 taxis efficiently. They deploy the incremental search approach, while in real world scenarios, this approach does not find all suitable taxis. The work in [4] considers the problem of on-line continual planning, in which additional ride requests may arrive while plans for previous ride-matching are being executed. DesTeller is a system for destination prediction based on trajectories considering passengers privacy [29], [30]. It uses Sub-Trajectory Synthesis (SubSyn) to solve data sparsity problem when query trajectories continue to non-terminal links. Another approach uses Hidden Markov Models in order to predict the future locations of moving objects [31]. Very recently, in the work [32], a multi-hop system was proposed to allocate more than one rider to a driver. The system has the ability to find itineraries for riders by means of optimally routing drive. This based on the assumption that each rider is served on a first-come, first-served basis.

In addition, estimating the travel time is a sub-problem of dynamic taxi ridesharing. It is a highly challenging problem because it deals with different factors such as traffic fluctuations, demand and supply, traffic signals, weather conditions and seasonal changes [12], [19]. One of the existing approaches proposes a real-time travel time estimation using sparse trajectories [14]. The requirements for this method is knowing the path for which a part of the path is not associated with previous values. One of the recent works proposes an approach for travel time estimation using large-scale taxi data with partial information [33]. The proposed model focuses on uncertainty in path choices. It infers the possible paths for each trip and then estimates the link travel times by minimizing the error between the expected path travel times and the observed path travel times. This model uses only the current time data and does not support historical data.

A survey over Trajectory Data Mining [34] divides the paradigm of trajectory data mining into sub-areas with different directions and our work focuses on the uncertainty in the trajectory planning. Different from the existing works under uncertainty sub-paradigm that focus on the uncertainty of the trajectory data, our work handles the uncertainty in the application context.

6.2 Data Uncertainty

Uncertainty in general is not a new problem and has been extensively discussed in the literature. A wide range of methods have been proposed to solve several types of problems in this area. The uncertainty of data may rise from data entry errors, integrated heterogeneous data sources, and the presentation style of data. In [16], ranking top k query results from large databases filled with uncertain data has been discussed. Another example for static uncertain data [35] makes use of a method called *probabilistic inverse ranking*. In the proposed model, the probability of each of the top K records is considered.

In spite of its applicability, the problem of uncertain spatio-temporal data has been rarely discussed [36]. One of the studies that takes this type of data into account is [37]. In that study, the problem of analysing a moving object's data has been modelled as a Markov chain and impossible states are pruned based on previous states of that object. Hence, the search area is reduced to possible states.

Uncertain data streams also have attracted researchers recently. For instance, Tran et al. [38] discuss conditioning and aggregation operations on uncertain data streams. Furthermore, some studies consider high-volume uncertain streams specifically. In [39], the proposed system employs probabilistic inference to generate uncertainty description

for its input (raw data), then a set of statistical methods are deployed to capture changes of uncertainty as data propagates through query operators. Our work differs with the above mentioned works on considering the uncertainties from the user decisions.

7 CONCLUSION

Despite recent active research efforts, dynamic ridesharing still remains a challenging problem. In this paper, we present the details of the TRIPS framework for dynamic taxi ridesharing. Our approach improves the results by considering the probability of users decisions, which is not previously addressed by the proposed systems in this area. We describe the details of three novel approaches including the DS, FS and IPFS, to search for the suitable alternatives. Unlike other state-of-the-art ridesharing solutions which use the IS approach, our approach focuses on finding the maximum number of taxis that match the query and then rank them based on certainty and closeness (i.e., the search once and rank strategy). We also reformulate the criteria for searching and ranking ridesharing alternatives to optimize the process. We conduct extensive experiments using a real-world dataset of taxi trajectories collected in Beijing, China to evaluate the proposed search approach. The experimental results show not only the efficiency and scalability of the proposed approach, but also the financial benefits brought by the approach.

There are several interesting directions for the future research. First, the nature of uncertainty in dynamic ridesharing is very complex. We plan to further investigate the modeling of different sources of uncertainty and analyze their impacts on taxi ridesharing. For instance, our observation of the real-world dataset shows that the validity of sensor readings are contaminated with a high level of uncertainty. Table 2 shows the speeds of taxis in the dataset that we have used. There are 51,147 records showing taxis traveling with high speed of 200 km/h or more, which are most likely due to errors in GPS readings for some taxis. Sensor uncertainty, which is likely to increase during the operation of the system, can result in incorrect and non optimal query results. Therefore, extending uncertainty factor is one of our future research directions. Finally, we also will target automating decision making and optimized stochastic planning for the cases in which, taxi or passenger(s) get missed due to the external factors such as predictable delays and traffic.

REFERENCES

[1] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 3, p. 38, 2014.
[2] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *European Journal of Operational Research*, vol. 223, no. 2, pp. 295–303, 2012.
[3] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proceedings of 29th International Conference on Data Engineering (ICDE 2013)*, Brisbane, Australia, April 2013, pp. 410–421.
[4] C. Manna and S. Prestwich, "Online stochastic planning for taxi and ridesharing," in *Proceedings of the 26th International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, Limassol, Cyprus, November 2014, pp. 906–913.
[5] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 7, pp. 1782–1795, July 2015.
[6] A. Shemshadi, Q. Z. Sheng, and W. E. Zhang, "A decremental search approach for large scale dynamic ridesharing," in *Proceedings of the 15th International Conference on Web Information Systems Engineering (WISE 2014)*. Thessaloniki, Greece: Springer, October 2014, pp. 202–217.
[7] A. Kleiner, B. Nebel, and V. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," in *Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI 2011)*, Barcelona, Spain, July 2011, pp. 266–272.
[8] W. Herbawi and M. Weber, "Modeling the multihop ridematching problem with time windows and solving it using genetic algorithms," in *Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012)*. Athens, Greece: IEEE, Athens, Greece 2012, pp. 89–96.
[9] K. Ghoseiri, A. Haghani, and M. Hamedi, *Real-time rideshare matching problem*. University of Maryland, 2011.
[10] N. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang, "Dynamic ride-sharing: A simulation study in metro atlanta," *Procedia-Social and Behavioral Sciences*, vol. 17, pp. 532–550, 2011.
[11] R. Baldacci, V. Maniezzo, and A. Mingozzi, "An exact method for the car pooling problem based on lagrangean column generation," *Operations Research*, vol. 52, no. 3, pp. 422–439, 2004.
[12] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: driving directions based on taxi trajectories," in *Proceedings of the 18th International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2010)*. San Jose, USA: ACM, November 2010, pp. 99–108.
[13] J. Krumm, R. Gruen, and D. Delling, "From destination prediction to route prediction," *Journal of Location Based Services*, vol. 7, no. 2, pp. 98–120, 2013.
[14] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2014)*. ACM, 2014, pp. 25–34.
[15] I. P. Levin, M. Mosell, C. Lamka, B. Savage, and M. Gray, "Measurement of psychological factors and their role in travel behavior," *Transportation Research Record*, vol. 649, pp. 1–7, 1977.
[16] M. A. Soliman, I. F. Ilyas, and S. Ben-David, "Supporting ranking queries on uncertain and incomplete data," *The VLDB Journal*, vol. 19, no. 4, pp. 477–501, 2010.
[17] R. Chrobok, O. Kaumann, J. Wahle, and M. Schreckenberg, "Different methods of traffic forecast based on real data," *European Journal of Operational Research*, vol. 155, no. 3, pp. 558 – 568, 2004.
[18] T. Pohlmann and B. Friedrich, "A combined method to forecast and estimate traffic demand in urban networks," *Transportation Research Part C: Emerging Technologies*, vol. 31, pp. 131–144, 2013.
[19] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, San Diego, USA, Aug. 2011, pp. 316–324.
[20] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah, "Sharek: A scalable dynamic ride sharing system," in *Proceedings of the 16th IEEE Intl. Conf. on Mobile Data Management (MDM 2015)*, Pittsburgh, June 2015, pp. 4–13.
[21] S. Abdel-Naby, S. Fante, and P. Giorgini, "Auctions Negotiation for Mobile Rideshare Service," in *Proceedings of the 2nd International Conference on Pervasive Computing and Applications (ICPCA 2007)*, Birmengham, UK, July 2007, pp. 225–230.
[22] G. Gidófalvi, G. Herenyi, and T. Bach Pedersen, "Instant Social Ride-Sharing," in *Proceedings of the 15th World Congress on Intelligent Transport Systems*, New York, NY, USA, November 2008, p. 8.
[23] J. Yousaf, J. Li, L. Chen, J. Tang, X. Dai, and J. Du, "Ride-Sharing: A Multi Source-Destination Path Planning Approach," in *Proceedings of the 25th International Australasian Joint Conference (AI 2012)*, Sydney, 2012, pp. 815–826.
[24] V. Dimitrieski, "Real-time carpooling and ride-sharing: Position paper on design concepts, distribution and cloud computing strategies," in *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Krakw, Poland, September 2013, pp. 781–786.
[25] C. Tian, Y. Huang, Z. Liu, F. Bastani, and R. Jin, "Noah: A Dynamic Ridesharing System," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*, New York, USA, June 2013, pp. 985–988.

[26] Y. Huang, R. Jin, F. Bastani, and X. S. Wang, "Large Scale Real-time Ridesharing with Service Guarantee on Road Networks," *Computing Research Repository*, 2013.

[27] M. Ota, H. T. Vo, C. T. Silva, and J. Freire, "STaRS: Simulating Taxi Ride Sharing at Scale," *IEEE Transactions on Bid Data*, vol. 3, no. 3, pp. 349–361, 2017.

[28] D. Zhang, T. He, Y. Liu, and J. A. Stankovic, "Callcab: A unified recommendation system for carpooling and regular taxicab services," in *Proceedings of the 2013 IEEE International Conference on Big Data (Big Data 2013)*. Santa Clara, USA: IEEE, October 2013, pp. 439–447.

[29] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Yu, and Y. Tang, "Desteller: A system for destination prediction based on trajectories with privacy protection," in *Proceedings of the VLDB Endowment*, vol. 6, no. 12. VLDB Endowment, 2013, pp. 1198–1201.

[30] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu, "Destination prediction by sub-trajectory synthesis and privacy protection against such prediction," in *Proceedings of 29th International Conference on Data Engineering (ICDE 2013)*, Brisbane, Australia, April 2013, pp. 254–265.

[31] D. Qiu, P. Papotti, and L. Blanco, "Future locations prediction with uncertain data," in *Proceedings of The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2013)*. Prague, Czech Republic: Springer, September 2013, pp. 417–432.

[32] N. Masoud and R. Jayakrishnan, "A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system," *Transportation Research Part B: Methodological*, vol. 106, pp. 218–236, 2017.

[33] X. Zhan, S. Hasan, S. V. Ukkusuri, and C. Kamga, "Urban link travel time estimation using large-scale taxi data with partial information," *Transportation Research Part C: Emerging Technologies*, vol. 33, pp. 37–49, 2013.

[34] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 29, 2015.

[35] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *Proc. of the 28th ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, Vancouver, BC, Canada, June 2008, pp. 673–686.

[36] G. Wu, Y. Ding, Y. Li, J. Bao, Y. Zheng, and J. Luo, "Mining Spatio-Temporal Reachable Regions over Massive Trajectory Data," in *Proceedings of the 33rd International Conference on Data Engineering (ICDE 2017)*, San Diego, CA, USA, April 2017, pp. 1283–1294.

[37] T. Emrich, H. Kriegel, N. Mamoulis, M. Renz, and A. Zufle, "Querying Uncertain Spatio-Temporal Data," in *Proc. of the 28th International Conference on Data Engineering (ICDE 2012)*, Washington, DC, USA, April 2012, pp. 354–365.

[38] T. Tran, A. McGregor, Y. Diao, L. Peng, and A. Liu, "Conditioning and aggregating uncertain data streams: Going beyond expectations," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1302–1313, 2010.

[39] Y. Diao, B. Li, A. Liu, L. Peng, C. Sutton, T. Tran, and M. Zink, "Capturing data uncertainty in high-volume stream processing," in *Proceedings of 4th Biennial Conference on Innovative Data Systems Research (CIDR 2009)*, Asilomar, USA, January 2009.

Ali Shemshadi is a Machine Learning Scientist in Complexia, Australia. His research includes a variety of fields such as Internet of Things, Supply Chain Management, Analytical Decision Making, and Machine Learning. He received his PhD from the University of Adelaide in 2016. He has also served on various conference committees and international journals in roles ranging from Committee Member and Reviewer, through to Program Chair. His teaching roles also include Associate Lecturer and Course Coordinator for a number of advanced courses at the University of Adelaide.

Quan Z. Sheng is a full Professor and Head of Department of Computing at Macquarie University, Australia. Professor Sheng holds a PhD degree in computer science from the University of New South Wales (UNSW), Sydney, Australia. His research interests include Internet of Things, Web of Things, big data analytics, service computing, distributed computing, and Internet technologies. He is the recipient of ARC Future Fellowship in 2014, Chris Wallace Award for Outstanding Research Contribution in 2012, and Microsoft Research Fellowship in 2003. He is the author of more than 330 publications. He is a member of the ACM and the IEEE.
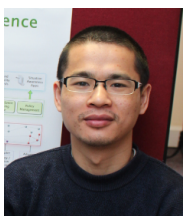
Yongrui Qin is a Senior Lecturer in School of Computing and Engineering, University of Huddersfield, United Kingdom. He obtained a PhD degree from University of Adelaide, Australia. His main research interests include Internet of Things, Web of Things, Semantic Web, data management, data mining and mobile computing. He has published more than 60 refereed technical papers.

Xiujuan Xu received the PhD degree in College of Computer Science and Technology from Jilin University in 2008. She is currently an Associate Professor in School of Software, Dalian University of Technology, China. Dr Xu worked at School of Computer Science, the University of Adelaide as a visiting scholar in 2016. Her research interests include data mining, intelligent transportation systems, recommender systems, and social network analysis. Dr Xu is the author of more than 30 publications.

Wei Emma Zhang is currently a postdoctoral research fellow in Department of Computing at Macquarie University, Sydney, Australia. Her research interests include the Internet of Things, text mining, data mining and knowledge base. She received PhD degree in computer science from the University of Adelaide in 2017. She has authored and coauthored more than 30 papers. She has served on various conference committees and international journals in different roles such as track chair, proceeding chair, PC member and reviewer. She is the member of the IEEE and ACM.

Jian Yang is a full professor and the research director at Department of Computing, Macquarie University. She received her PhD from the Australian National University in 1995. Before she joined Macquarie University, she worked as an associate professor at Tilburg University, Netherlands (2000-2003), a senior research scientist at the Division of Mathematical and Information Science, CSIRO, Australia (1998-2000), and a lecturer at Department of Computer Science, The Australian Defence Force Academy, University of New South Wales (1993-1998). Her research interests include interoperable systems, data/process integration, social networks, big data analytics, and smart cities.