IIP: An Information Platform for Intelligent Transportation System and Its Applications

BY

SHUO MA
B.S., Beijing University of Posts and Telecommunications, 2008

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2014

Chicago, Illinois

Defense Committee:
Professor Ouri Wolfson, Computer Science, Chair and Advisor
Professor Philip Yu, Computer Science, Co-advisor
Professor Prasad Sistla, Computer Science
Professor Jane Lin, Civil and Materials Engineering
Dr. Dr. Bo Xu, Nokia

This dissertation is dedicated to my wife, Zhongtao Xie, without whom I would have finished my Ph.D. a little earlier and missed tons of happiness.

ACKNOWLEDGMENTS

First I owe a great debt of gratitude to my advisors, Professor Ouri Wolfson and Professor Jane Lin. They have provided support and guidance since my admission to the program. I could never accomplish this dissertation without their help.

I would also like to express my sincere thanks to other members of my committee: Professor Philip Yu, Professor Prasad Sistla, and Dr. Bo Xu for their valuable comments and suggestions to my dissertation.

I am also grateful to my family, colleagues, and friends for their support and encouragement. Finally, I would like to thank the National Science Foundation and the Computer Science Department for their generous funding.

SM

Contribution of Authors

Chapter 1 represents a published manuscript [75] and Chapter 2 represents a published manuscript [76]. For both manuscripts, I was the first author, and my advisors, Dr. Wolfson and Dr. Lin, contributed to the writing of the manuscript. Chapter 3 represents a published manuscript [77] for which I was the primary author and major driver of the research. Dr. Yu assisted me in the experiments and the figures. Dr. Yu and Dr. Wolfson contributed to the writing of the manuscript. Chapter 4 represents a published manuscript [74] for which I was the first author and Dr. Wolfson helped me with writing. Chapter 5 represents an unpublished report for which I was the primary author, and Dr. Wolfson, Dr. Xu and Roland Varriale contributed to the writing. Chapter 6 represents a submission for which I was the first author, and Dr. Wolfson and Dr. Xu helped me with the writing and experiments. I anticipate that this work will ultimately be published as part of a co-authored manuscript.

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

ITS	Intelligent Transportation System
IIP	Intelligent transportation system Information Platform
GSM	Global System for Mobile communications
GPRS	General Packet Radio Service
ESR	Event Schemas Generator
P2P	Peer-to-Peer
UPnP	Universal Plug and Play
UDDI	Universal Description and Discovery and Integration
APP	APPlication
OD	Origin and Destination
T-Share	Taxi-Share
TDOTRP	Total Distance Optimization Taxi Ridesharing Problem
TSPTW	Travelling Salesman Problem with Time Window
KNN	K Nearest Neighbor
DARP	Dial A Ride Problem
SP	Slugging Problem
GSP	Generalized Slugging Problem
CSP	Capacitated Slugging Problem
DBSP	Delay-Bounded Slugging Problem
DBCSP	Delay-Bounded and Capacitated Slugging Problem
VANET	Vehicular Ad-hoc NETwork
UGC	User Generated Content
MANET	Mobile Ad-hoc NETwork
GIS	Geographical Information System

LIST OF ABBREVIATIONS (continued)

VGI	Volunteer Geographical Information
VTIS	Volunteer Traveler Information System
UPDetector	Unparking/Parking activities Detector
SMS	Short Message Service

SUMMARY

This dissertation presents an information platform for Intelligent Transportation System (ITS) applications with a focus on trust management issues, and three example ITS applications, namely ridesharing, Volunteer Transportation Information System (VTIS) and parking/unparking activities detection using smart phones.

The information platform, referred to as the ITS Information Platform (IIP) [75] hereafter, is motivated by relieving ITS application developers from implementing communication and data management components which are necessities but not central to the application functionality. In other words, IIP, residing as a middleware between operating systems and ITS applications, provides primitives to application developers for communication and data management needs. IIP consists of two parts, namely the *Cloud Component* and the *Client Component*. The IIP Cloud Component provides canonical publish/subscribe (pub/sub) functions to both traffic management facilities and mobile nodes, e.g. vehicles, smart phones. The IIP Client Component provides mobile nodes with pub/sub functions which allow them to communicate with the IIP Cloud Component as well as with each other. Via leveraging heterogeneous data sources and various communication mechanisms in the ITS environment, IIP supports a wide variety of ITS applications.

A pressing concern with IIP is trust management: should a subscriber trust the publications she has received? What IIP can do to help subscribers manage risks of being exposed to false information or tricked by malicious publishers? Towards this end, existing works on trust management for ITS is surveyed [76].

IIP is motivated to help ITS application developers. Therefore, it is in our interest to investigate some emerging yet promising ITS applications in order to better understand the needs of ITS

application developers. As case studies, we investigate three ITS applications, namely ridesharing, the VTIS and the parking/unparking activities detection using smart phones.

Ridesharing helps alleviate many existing major transportation problems, such as traffic jams, find parking spaces, hard to hail a taxi during rush hours. These problems are perennial headaches for cities, especially those with a large population, and affect the environment, the economy, and more directly average people's daily lives. We treat ridesharing as a constrained optimization problem. A variety of constraints can be considered when modeling a specific ridesharing application. For example, capacity constraints limit the maximum number of riders on a vehicle at the same time; spatial constraints define the Origin-Destination (OD) pair of a trip; temporal constraints define the desirable time windows in which the trip should take place; and monetary constraints provide incentives for riders and drivers to participate in ridesharing.

The two major objectives of ridesharing are efficiency and effectiveness. Efficiency concerns about how fast on average each trip request, i.e. query, is processed, either assigned to a vehicle or denied for ridesharing. Efficiency becomes a more acute and bigger concern than effectiveness for dynamic ridesharing problems in which queries arrive in real-time instead of being known in advance. We propose and develop a taxi-sharing system called T-Share that accepts taxi passengers' real-time ride requests sent from smart phones and schedules proper taxis to pick up them via ridesharing, subject to time, capacity, and monetary constraints. The T-Share system is built based on a mobile-cloud architecture. Taxi riders and taxi drivers use the taxi-sharing service provided by the system via a smart phone APP. The Cloud first finds candidate taxis quickly for a taxi ride request using a taxi searching algorithm supported by a spatio-temporal index. A scheduling process is then performed in the Cloud to select a taxi that satisfies the request with minimum

increase in travel distance. We evaluate the T-Share system with extensive experiments to validate its effectiveness, efficiency and scalability.

Effectiveness of ridesharing concerns about how much benefits, such as decrease in travel distance, ridesharing can bring. And we are interested in quantifying such benefit and find the theoretical bound of the defined benefit function with the presence of various constraints. As an example of such efforts, we analyze the slugging form of ridesharing [74], where passengers instead of drivers change their route (i.e. a passenger needs to walk to the origin of the driver, share the ride with the driver, get off at the driver's destination and then walk back to his/her destination). We formally define the slugging problem and its generalization. We provide proofs of their computational time complexity. For the variants of the slugging problem that are constrained by the vehicle capacity and travel time delay, we prove NP-completeness and also propose some effective heuristics. In addition, we discuss the dynamic slugging problem.

Volunteer Traveler Information System [112] is another application proposed for the IIP. It aims to provide travelers real time transport related information, such as traffic conditions, accidents, bus/train delays, parking spaces availability, etc. Such information has paramount values for travelers on the road. Nowadays, travelers often get limited types of real-time transport information (mostly traffic condition info. on major roads like highways and arterials) from large Internet companies like Google or traffic-oriented websites like traffic.com. A potentially ideal way to get more comprehensive and accurate real-time transport information is crowd sourcing, i.e. gleaning data from a large number of travelers. This approach becomes practically feasible as currently most travelers carry mobile devices which are capable of reporting such information via wireless communication. Thus, a real-time travel information notification system such as the VTIS is of great value and in an urgent need. The VTIS provides two essential functions for travelers, i.e.

publish/subscribe. Specifically, the publish function of the VTIS allows a traveler to easily report a transport related event via the cell phone immediately whenever she witnesses an interesting event no matter while driving, riding a bike/bus/train, or walking. Symmetrically, the subscribe function of the VTIS allows a traveler to subscribe to the specified information of interest via the phone, and the VTIS will automatically notify the traveler whenever one of her subscriptions gets satisfied, i.e. some other people have reported information of her interest. We envision that the VTIS brings pragmatic merits to both the general public and transportation authorities. On one hand, it greatly conveniences average travelers by feeding them more timely and accurate customized travel information. On the other hand, it also provides a complementary mass-powered transport information source for authorities. As a result, authorities can improve their existing services by exploiting the data gleaned by the VTIS.

The third ITS application we consider here is the detection of parking/unparking activities using smartphones. Real-time information about vacant parking spaces is of paramount value in urban environments. One promising approach to obtaining such information is participatory sensing, i.e. detecting parking/unparking activities using smartphones. We introduces and describes multiple indicators, each of which provides some inconclusively clue for a parking or an unparking activity. As a result, we proposes a probabilistic fusion method which combines the output from different indicators to make more reliable detections. The proposed fusion method can be applied to inferring other similar high-level human activities that involve multiple indicators which output features asynchronously. The proposed indicators and the fusion method are implemented as an Android App called UPDetector. Via experiments, we show that the UPDetector is both effective and energy-efficient.

Chapter 1

IIP: An Event-Based Platform for ITS Applications

1.1 Introduction

Envisioning that there are many common data management and communication elements shared by various prospective IntelliDrive[SM] [2] applications, we propose the *ITS Information Platform (IIP)* as a common data management and communication services platform facilitating and easing applications development. The motivation of IIP comes from the fact that there is a lack of a generic platform, which provides data management and communication support capability in a distributed, heterogeneous data environment like IntelliDrive[SM]. As an example, consider how an application helps a driver who is interested in the current and expected traffic speed on the Kennedy Expressway receive the information. The application first identifies the relevant information that contains or may be used to infer traffic speed on the Kennedy Expressway. Traffic reports providing exact speeds are the ideal source. A picture or a video clip of the expressway may be useful as well. Special events announcements about sports events, roadwork, accidents, and extreme weather warnings are also relevant. The issue is that such information may exist in different forms, in a web post or personal text communication, on the Internet or a hand-held devices, etc. And the solution to locating the data dictates the approach to accessing the data.

Thus a common data management and communication services platform like IIP is of great need to ease the burden on the application developers and users from having to deal with issues such as finding the data discussed above. We argue that the publish/subscribe (pub/sub) paradigm [41] is an appropriate fundamental building block for IIP. Publish/Subscribe is an asynchronous communication paradigm that provides spatiotemporally loosely-coupled connection between communicating parties. Moreover, it is multicasting in nature since data producers can send the

same data to multiple consumers with a single operation. As a result, the asynchrony and one-to-many characteristics make the publish/subscribe system more suitable than other communication paradigms, e.g. query-response and remote procedure call, for distributed information dissemination applications which are prevalent in the ITS environment. We describe the components, their functionality, and the architecture necessary for the pub/sub based IIP platform.

1.2 Architecture and Components

At the highest level, IIP is composed of two components, one residing on the cloud, i.e. the infrastructure network, which is referred as the *IIP Cloud Component*; and the other residing on the clients, which is referred as the *IIP Client Component*. The IIP Cloud Component provides canonical publish/subscribe functions to both traffic management facilities and mobile nodes. The IIP Client Component provides mobile nodes with pub/sub functions which allow them to communicate with the IIP Cloud Component as well as with each other.

Following the conventions in the pub/sub literature, we use the term event for any useful information and publisher and subscriber respectively for producer and consumer of the event in the rest of this document. Figure 1 shows the overall architecture of IIP whose components are represented by dashed incarnadine boxes.

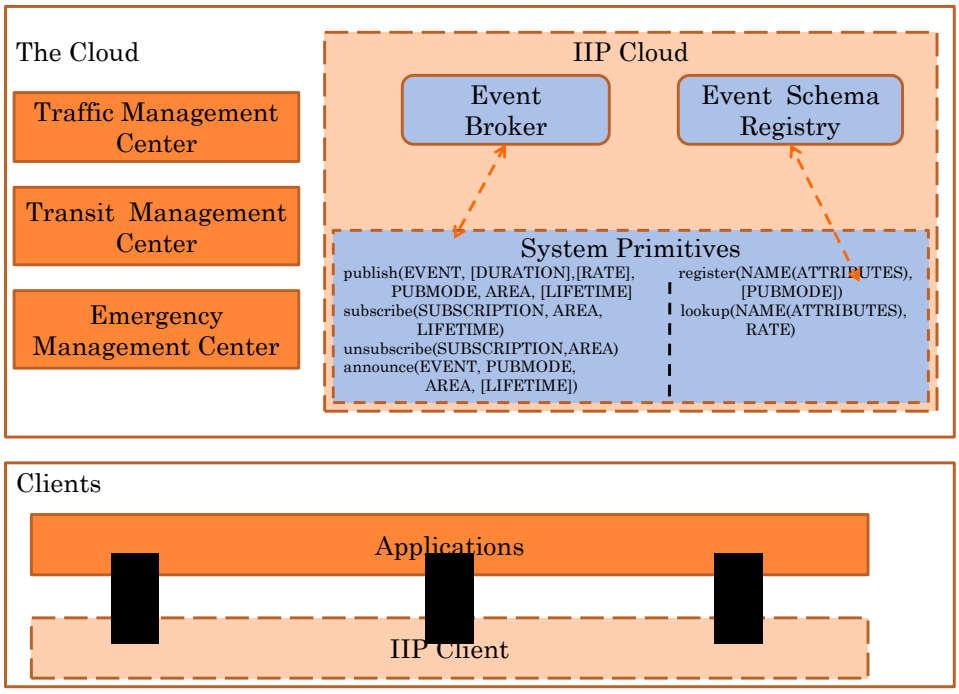


Figure 1 ITS Information Platform architecture

The IIP Cloud Component essentially consists of two building blocks, namely the *Event Broker* and the *Event Schema Registry (ESR)*. The Event Broker is the kernel function of large scale infrastructure-based publish/subscribe systems. It basically performs "store and forward" function to route events from publishers to subscribers. Upon the arrival of new events, the Event Broker sends the events to all the subscribers whose subscription matches the new event.

The Event Schema Registry provides a repository where publishers can find event schemas of interest. The System Primitives provide the open interface for applications running on either mobile nodes or management facilities to access the IIP Cloud Component. More detailed descriptions of the Event Schema Registry and the Event Broker will be introduced soon.

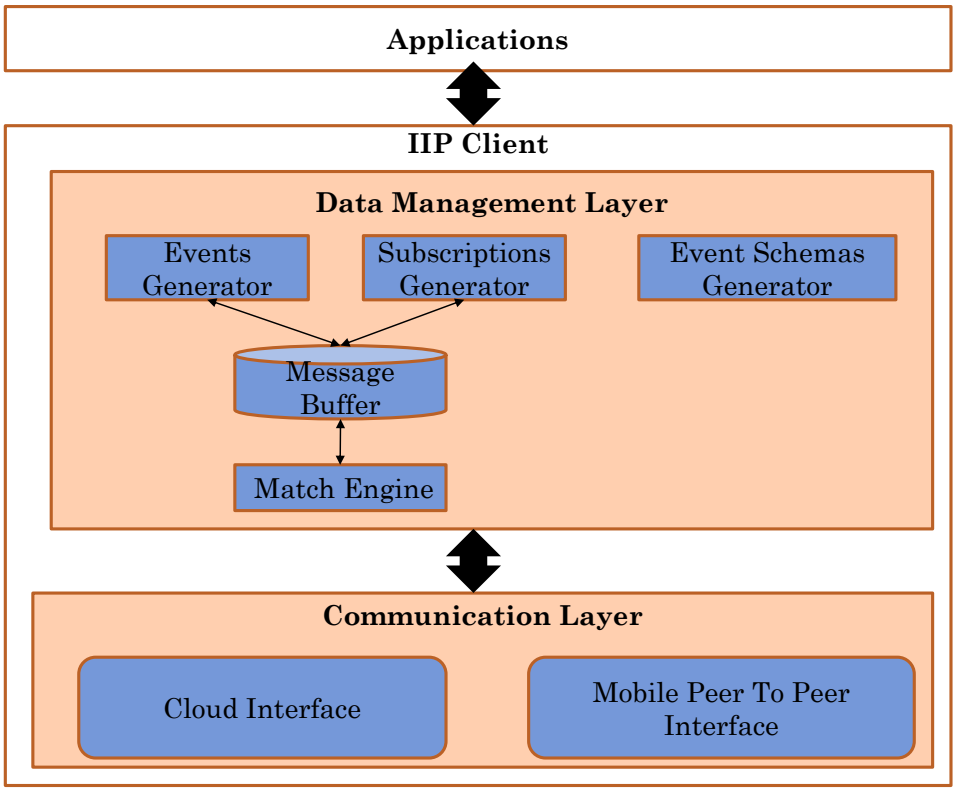


Figure 2 IIP Client architecture

Figure 2 shows the architecture of the IIP Client Component. It employs a two-layer structure. The upper level is the *Data Management Layer*. It provides applications with a uniform data abstraction and manipulation tool compatible with the IIP Cloud Component. Specifically, the *Event Schemas Generator* allows mobile nodes to produce event schemas to be registered to the ESR. The *Events Generator* and the *Subscriptions Generator* control the generation of events and subscriptions respectively. The Event Broker module provides the function of brokering events and is implemented separately by interfaces defined in the underlying communication layer. The Match Engine implements a generic matching mechanism between events and subscriptions. Note that the IIP Client Component does not provide a schema discovery function, thus mobile nodes cannot learn event schemas of interest from anywhere else but the Event Schema Registry in the Cloud.

The lower level is the *Communication Layer*, which supports different communication paradigms. Currently two paradigms are considered. The *Cloud Interface* allows mobile nodes to

access the IIP Cloud Component through Internet access points, e.g. Wi-Fi hotspots, cellular base stations or other road side units. Existing standard cellular system, e.g. GSM and GPRS, and works on Internet access protocols, e.g. fast Wi-Fi access protocols described in [40] are sufficient for realizing the Cloud Interface. Notice that the Cloud Interface cannot support pub/sub function for purely mobile peer to peer applications in which the communication infrastructure does not exist, or is not fast enough. As an example, consider the application where the right-of-way is distributively maintained among vehicles, pedestrians and cyclists at a road intersection without traffic lights or human coordinators. In such an application, mobile nodes approaching the intersection should coordinate with each other directly via short range high-bandwidth channels, since a cloud solution probably cannot guarantee the necessary response time.

Therefore we propose the *Mobile Peer to Peer (P2P)* Interface as a complementary communication method. It implements the pub/sub system purely via inter-node communication in a mobile P2P network (vehicles, smart-phones and other pedestrian/bicyclist devices), regardless of whether or not the mobile P2P network is connected. In other words, we assume that the Data Management Layer on IIP Client is not concerned with connectivity issues in the Mobile P2P network, and the communication layer will employ the appropriate Delay Tolerant protocols to overcome connectivity problems in the this network.

1.3 Primitives for Information System for ITS

1.3.1 The Event Schema Registry

In this section, we present the primitives with their associated parameters provided by the Event Schema Registry. And we classify those parameters into two categories, i.e. mandatory parameters and optional parameters. Mandatory parameters are semantically required, i.e. they are indispensable for the purpose of providing the necessary semantics of the primitives. Optional parameters are not compulsory and used as “hint information” by IIP to enhance the performance of the system. Notice the same parameter may be mandatory to one primitive while optional to

another. The decision is dependent on the specific semantics of the primitive. We will indicate whether or not a parameter is optional for a primitive in proper timings in the rest of the section.

The Event Schema Registry is implemented as a central directory which enables publishers to register their event schemas and provides schema search functionality to subscribers. From an application's point of view, it provides the following two essential primitives (optional parameters are enclosed by brackets):

- *register(EVENT_SCHEMA)*
- *lookup(EVENT_SCHEMA, RATE)*

1.3.1.1 The register Primitive

Each publisher registers an event schema with the Event Schema Registry by invoking the register primitive. The *EVENT_SCHEMA* parameter is formally represented in the format of SCHEMA_NAME(ATTRIBUTES), where SCHEMA_NAME and ATTRIBUTES represents the name and the attributes of the event to be registered by the publisher respectively. An attribute can be a primitive data type, such as a string, a numeric field, a spatiotemporal data structure such as a point, region, hour, day, or a complex data structure, such as an XML document. For a complete example of a schema, consider: "BUS LOCATION" with attributes <ROUTE_NO, RUN_NO, TIMESTAMP, LOCATION-COORDINATES>.

1.3.1.2 The lookup Primitive

Each subscriber discovers the event schemas of interest from the Event Schema Registry by invoking the *lookup* primitive. The *EVENT_SCHEMA* parameter has the same format and semantics of that in the register primitive. We treat each call to the lookup primitive as an instantaneous rather than a continuing query. Consequently, the Event Schema Registry saves considerable memory space by not keeping track of the history of received lookup calls.

Each call to the *lookup* primitive from an application is passed to the IIP Client Component, which in turn sends the call to the IIP Cloud via the Cloud Interface. The Event Schema Registry

returns the IIP Client Component a list of all event schemas that match the *EVENT_SCEHMA* parameter of the call. Here the matching is performed using both schema matching techniques [52] and ontology matching techniques [42].

Prior to displaying the returned list of event schemas from the ESR to the application, the IIP Client Component ranks the schemas based on some measures, such as reputation of the publisher of the event schema if a reputation system is implemented. Then the subscribers may choose schemas on which they would like to write subscriptions.

The *RATE* parameter is introduced such that subscribers are constantly kept posted about new schemas of interest registered with ESR. To understand why, consider the following scenario. Suppose a driver is interested to traffic speed information on his way to home. So s/he invokes a call to the *lookup* primitive, where the *EVENT_SCHEMA* parameter equals "traffic speed"(TRAFFIC_SPEED, FLOW, ROAD_NAME, TIME), where "traffic speed" is the name of the schema. Further suppose s/he receives the returned event schema with attributes <SPEED, TRAFFIC_FLOW, LOCATION, TIMESTAMP> from the ESR. Assume later a new schema with attributes <VELOCITY, FLOW, ROADLINK_ID, TIME> is registered, which also matches the driver's interest. However, the driver has no way of knowing about it unless s/he invokes the lookup primitive with the same parameter again later. Clearly, some kind of "update" service is desirable from users' perspectives and that is why parameter *RATE* is introduced. Here is how it works. For each call C to the *lookup* primitive, the IIP Client retransmits C to the ESR at the frequency indicated by the *RATE* parameter given in C. As such, the user receives updated information on relevant schemas periodically. In addition, we assume a default valid time for each call to prevent the IIP Client Component from retransmitting the call forever.

1.3.2 The Event Broker

The Event Broker serves as the message "bridge" connecting publishers and subscribers. It provides the following primitives:

- *publish(EVENT, EVENT_SCHEMA, [PUBMODE], [AREA],[LIFETIME])*
- *subscribe(PREDICATE, EVENT_SCHEMA, [PUBMODE] AREA, LIFETIME)*
- *unsubscribe(PREDICATE, EVENT_SCHEMA)*
- *announce(EVENT, EVENT_SCHEMA, [AREA], [PUBMODE],[LIFETIME])*

Next we first give precise definitions for all primitive parameters and explain the semantics of each primitive, and then we discuss how the Event Broker is implemented.

1.3.2.1 The publish Primitive

Each node invokes the *publish* primitive when it needs to publish any events. The EVENT parameter represents an instance record of the event schema given by the EVENT_SCHEMA parameter.

The *PUBMODE* parameter: Given an subscribe call from an application, the IIP Client Component may need a mechanism to intelligently decide which action to take, i.e., to deliver the subscription to the IIP Cloud Component via the Cloud Interface or to disseminate the subscription using the Mobile Peer to Peer Interface, or both. Thus the *PUBMODE* parameter is introduced as the basis on which the IIP Client Component makes such a decision.

Note that in the case of the PUBMODE parameter is absent, the IIP Client Component simply tries both actions. In other words, PUBMODE does not affect the semantics of the register primitive but only helps the IIP Client Component improve the performance by saving unnecessary communication cost. Therefore we treat PUBMODE as an optional parameter to the register primitive.

Specifically, the PUBMODE parameter indicates the mode in which a publisher publishes its events of the given SCHEMA. More specifically, the value of *PUBMODE* can be (i) *mobile*, meaning publishing events through the embedded Mobile Peer to Peer Interface, i.e. disseminating events among peers; (ii) *cloud*, meaning publishing events to the IIP Cloud Component via the embedded Cloud Interface, i.e. transmitting events to the Event Broker; or (iii) *mixed*, meaning publishing events in both mobile and cloud modes. For an example of publishers using different

values for *PUBMODE*, consider the application described in [66] where private cars disseminate events reporting the witness of a fleeing car driven by some criminals and the patrolling police cars reap such events. In this case, the private cars are using mobile mode to publish events. And the police cars may use cloud mode to report the collected events to the headquarter of local police department where a chasing and hunting plan is made.

Notice the value of the *PUBMODE* parameter makes no indication about the mobility status of the publisher. For example, a publisher who uses mobile publishing mode may be a moving probe car or a fixed roadside sensor. On the other side, some publishing modes may do not make sense provided the mobility status of the publisher. For instance, it is not likely that a desktop user sitting in the office will publish traffic speed events (assumedly obtained by the user from another source other than IIP) using the mobile mode. However, even though s/he does, the publishing action will not take any effects.

The *AREA* and *LIFETIME* Parameters: The *AREA* parameter specifies the region where the intended recipients of the published events should locate. By intended recipients, we refer to the anticipated receivers of the events from the publisher's point of view. Thus though nodes outside of the region defined by the *AREA* parameter may also receive the events, they are considered only as brokers who relay the events. Notice we expect that most of the time the *AREA* parameter is used by the Mobile Peer to Peer Interface, i.e. when the *PUBMODE* parameter has a value of *mobile* or *mixed*. In other words, any specific implementation of Mobile Peer to Peer Interface is supposed to support for geocasting, i.e. delivering events to a group of nodes within certain geographical area. However, the AREA parameter may be used when *PUBMODE* equals *cloud* as well.

Since it is hard for users to clearly state the value of the *AREA* parameter using expressions, the IIP Client Component will provide a Graphic User Interface (GUI) for the purpose of specifying *AREA*. More specifically, the GUI will display the real time road network around the node, which typically consists of nearby parallel lanes in both directions. And it provides shapes, such as circle,

rectangle etc., that are used to select areas on the displayed road network. For an example of specifying *AREA* using the GUI, consider Emergency Electronic Brake Light (EEBL) application where a car disseminates an event to downstream vehicles when it detects the driver suddenly brakes heavily. In this case, when a car publishes a hard brake event, the value of the *AREA* parameter can be specified by selecting the rectangle area between the current location of the car and certain point on the same lane behind the car.

The *LIFETIME* parameter indicates how long the event is to be valid in the network. An invalid event will be discarded by all its carriers.

PUBMODE, *AREA* and *LIFETIME* are all optional parameters to the *publish* primitive because they are simply "performance hints". In other words, they affect performance, but not the semantics of the application.

1.3.2.2 The subscribe/unsubscribe Primitive

Each node invokes the *subscribe* primitive when they need to subscribe to events of interest. The *AREA* parameter and the *LIFETIME* parameter have the similar semantics of their counterparts in the *publish* primitive. However, here they are mandatory to the *subscribe* primitive because they are necessary for subscribers to precisely express what the events of interest are. Without them, the semantics of a subscribe primitive become paralyzed. For example, in the aforementioned EEBL scenario, a subscriber cannot express that s/he is only interested in hard brake events published in last 30 seconds by cars ahead of it within 1 mile without the *AREA* and the *LIFETIME* parameter. At any time, a subscriber can call the *unsubscribe* primitive to revoke any previous subscription.

The *PREDICATE* Parameter: The *PREDICATE* parameter represents a concrete subscription of certain subscriber. A subscription is either (i) an *Event Filter Predicate* or (ii) an *Event Pattern Predicate*.

An Event Filter Predicate is a conjunction of multiple predicates over a single event. Each predicate is a binary predicate over an attribute of the event, returning either TRUE or FALSE. An

Event Filter Predicate is matched against individual events and a match is found if an event makes the evaluation of the predicate return TRUE. For example, consider a driver subscribes to weather forecast events which are published every 15 minutes. The attributes of the weather events are <TIME_INTERVAL, TEMPERATURE, WIND_STRENGTH, PRECIPITATION, ULTRAVIOLET_INDEX>. An example of an Event Filter Predicate is "notify me if there is a predicted rain with the precipitation over 100 millimeters coming in an hour". Formally, the predicate can be written as "overlap (TIME_INTERVAL, next_hour) and PRECIPITATION>100 mm". And the predicate will be evaluated on each individual weather event to filter out all events whose TIME_INTERVAL and PRECIPITATION value satisfy the above conditions.

In contrast to an Event Filter Predicate, an Event Pattern Predicate is often defined and evaluated across multiple events of the same event schema. For instance, again consider a driver subscribes to the aforementioned weather forecast events. "Notify me if the temperature drops 10 degrees within 30 minutes" is an example of an Event Pattern Predicate since it can only be evaluated on at least two events. Another example of an Event Pattern Predicate in the above context could be "notify me if when the temperature reported by the weather forecast events has monotonically increased for two hours". In this case, the predicate has to be evaluated based on all-weather events in a temporal window of two hours.

We utilize the language models described in [35] to formally define the Event Pattern Predicates. The language model essentially consists of a set of formally defined language operators, such as projection, union, conditional sequence, iteration, etc. Given the defined operators and expressions, matching an Event Pattern Predicate against an event stream can be done using extended finite state automata model.

1.3.2.3 The announce Primitive

The *publish* and *(un)subscribe* primitives provide the basic semantic of a pub/sub system, more primitives are required by developers to construct various applications. An additional primitive we

considered is the *announce* primitive. All parameters in the *announce* primitive have the same meaning as the counterparts defined in the *publish* primitive. The difference between the *announce* primitive and the *publish* primitive lies in that announced events will be delivered to all subscribers located in the region defined by the *AREA* parameter regardless of whether or not their subscriptions match the events. Thus clearly the *AREA* parameter is mandatory, i.e. semantically required by the *announce* primitive. The *PUBMODE* and the *LIFETIME* parameters remain optional due to the same reason discussed in the *publish* primitive. The best proper scenarios for the *announce* primitive are emergency notification applications, such as delivering warning, evacuation or rescue events to drivers in some affected urban area after a devastating earthquake.

1.3.2.4 Implementation of the Event Broker

Recall that we define three different modes, namely *mobile*, *cloud* and *mixed*, for the *PUBMODE* parameter. This means that for each event schema, there are 3 modes to transmit the events and subscriptions of the schema in a network. Therefore given an event schema, there are 9 different scenarios in terms of how the events and subscriptions are transmitted respectively. For each of those 9 scenarios, the matching between events and subscriptions can be implemented using the Event Broker modules in the Cloud Interface and the Mobile Peer to Peer Interface. Next we describe how the Event Broker module is implemented in each of the two interfaces.

The most straightforward way to implement the Event Broker in the Cloud Interface is to implement it in a centralized approach, like the Event Schema Registry does. However, event and subscription submission (including subscription updates and cancelations) rate is much higher than event schema registering rate for a large scale heterogeneous network. Thus the single server implementation is not reliable or feasible. A more promising approach is to implement the Event Broker as a set of autonomous, interconnected distributed servers. The servers form a flat overlay network and forward events or/and subscriptions among themselves to make sure each event is matched against each active subscription in the overlay network. We employ the forwarding

algorithm described in [9] to implement the Event Broker in the Cloud Interface. More details about the applied forwarding algorithm can be found in [9].

There are many alternatives to implement the Event Broker module in the Mobile Peer to Peer Interface. For example, we can define that each node stores its subscriptions locally and only publishes events among peers. Even for this simple strategy, many rules, such as which peers can be brokers of which events and when the brokers retransmit received events, needs to be specified in order to optimize the system performance in terms of message overhead, delivery ratio, etc. Proposing a comprehensive and optimized implementation of the Event Broker in the Mobile Peer to Peer Interface is beyond the scope of this document and thus we do not discuss it further.

1.4 Related Work

The Event Schema Registry in IIP is related to work on service discovery. Currently existing service discovery protocols generally fall into two categories in terms of the way in which service information is managed. One approach is to broadcast service information through the entire network either using a gossip algorithm or maintaining a multicast tree. Salutation [9] by IBM and Universal Plug and Play (UPnP) [12] by Microsoft are representative members of this category. We believe such an approach suffers from a high cost for broadcasting or tree construction and maintenance in highly mobile networks, and thus is not suitable for IIP. An alternative approach taken by other protocols such as Universal Description and Discovery and Integration (UDDI) [10] for web services and Jini [11] is to maintain a central directory to store service information. The Event Schema Registry component of IIP also takes a central-directory approach, and can indeed use a web service system for implementation.

The pub/sub literature is also relevant to IIP. There are basically two types of publish/subscribe systems, namely topic based and content based [56]. In topic based systems, users express interests by simply joining a group defined by a central subject. Whereas content based systems provide

much more flexibility in expressing users' interests by allowing users to specify predicates over a set of attributes. As a result, arbitrary queries over the events content can be easily posed by users.

The pub/sub system in IIP employs a content-based approach. In contrast to traditional pub/sub systems [43, 103, 128] which are cloud-based, IIP supports an arbitrary combination of mobile peer-to-peer and cloud publications and subscriptions. In terms of purely mobile P2P implementations, a few pub/sub systems have been proposed. For example, [81] proposes a publish/subscribe implementation for MANETs. In the implementation, subscriptions are only deployed locally due to the fact that the paths utilized by the subscription forwarding strategy in server overlay networks quickly become stale in a mobile environment. When receiving an event from a neighbor, a mobile node matches the event to its own subscriptions and broadcasts the event, as long as it is still valid given the current location and time. Triantafillou and Aekaterinidis [108] present a different approach to support P2P applications via building a pub/sub middleware over a structured P2P network such as Chord [106]. But that solution is again restricted to a static P2P network.

1.5 Discussion

The development of ITS applications imposes the need for a platform which provides generic data management and communication functionalities. To meet the need, we proposed IIP as a way to facilitate the development of a variety of prospective ITS applications. We outlined the architecture of IIP and discussed the implementation of its two major components, namely, service discovery and publish/subscribe.

Several open issues require further exploration. Here we do not propose any specific mobile peer to peer protocol for the Communication Layer of the IIP Client Component. Although some existing research is concerned with this issue [56, 98], there still remain many interesting problems when attempting to apply those approaches to the ITS environment. For instance, in order to

expedite the information dissemination in the mobile P2P network, in additional to short range communication methods, wide area wireless networks, e.g. the cellular network, can be utilized. More specifically, to improve performance both publications and subscriptions can be disseminated. When a subscription and publication are matched at a node, since the net id of the subscriber is available, the event can be directly delivered to the subscriber via the cellular network. Thus, a subscriber may receive the event either via the mobile P2P network, or via the cellular network, whichever comes first. A similar approach was taken in [116] for matching meta-data and queries in a vehicular network. Also, how to make the pub/sub protocol adapt to various combinations of applications and network conditions remains an open question.

Chapter 2

Trust Management for Intelligent Transportation System

2.1 Introduction

In recent years, ITS (Intelligent Transportation System) has drawn increasing professional attention from researchers in academia and industry companies as well as official authorities, and has been considered the next life-changing technological revolution. The prospect of ITS promises a variety of applications, including safety applications, crowd-sourcing applications, entertainment applications, etc. Many prototypical applications [65, 66, 107] have been proposed. All of these proposals focus mainly on the implementation of application-specific functionalities, yet they all overlook a fundamental issue: namely, trust management.

Trust is a pervasive concept used in many disciplines like sociology, economy, psychology, computing, etc. [61]. We consider the semantics of "trust" only in the field of distributed systems and networking. Specifically, in traditional online e-commerce environments such as EBay, Amazon, the beta reputation system [60], etc., trust management, roughly speaking, refers to the management of the trustworthiness of relationships among entities. For the sake of simplicity, in a trusted relationship in which entity A trusts entity B, we refer to A as the trustor and B as the trustee. In a trust management scheme for online communities, the direct consequence of a distrusted relationship typically is to prevent any interaction between the two involved entities from happening.

In wireless networks like VANETs, ITS, etc., the concept of trust inherits its old interpretation from online e-commerce environments and also extends to the trust of an entity to data. For instance, in a crowd sourcing application, entities need to make trust decisions on received messages. We refer to the former interpretation of trust as *entity trust* and the latter interpretation as *data trust*.

One way to implement data trust management is to employ entity data trust management. As shown in Figure 3, data trust management can be layered above entity trust management. It utilizes entity trust values generated by the entity trust management layer to output data trust values. Data trust values are converted to binary decisions via comparing to a trust threshold value. Those decisions are in turn fed back to the entity trust management layer for updating entity trust values. To illustrate the above relations, consider a crowd sourcing application as an example. When an entity needs to evaluate the authenticity of a received message, it can use its trust value for the message's generator as the ground for the calculation of the trust value for the message. The resulted trust value is then converted to a trust decision. Such a decision will later be used to update the entity's trust value for the message's generator. Note that the dependence on entity trust management is not indispensable for data trust management implementations. Some proposed data management schemes [53, 95] only exploits the fact that information is often redundant in ITS to deal with data trust issues.

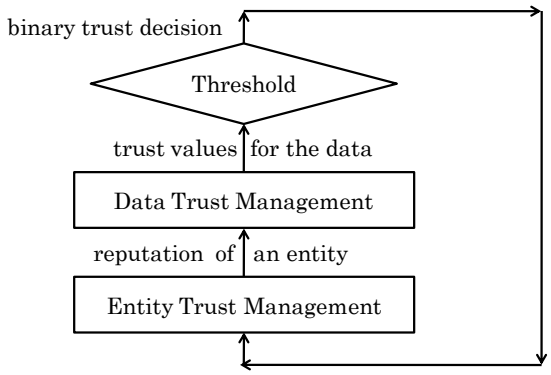


Figure 3 Potential dependence between entity and data trust management

2.2 Concepts of Trust Management

In this section, we examine the unique properties of trust in the context of Intelligent Transportation System.

Trust is *partially transitive*. On one hand, transitiveness implies that trust can be acquired either directly or indirectly. Direct trust is always earned via individual experiences. Indirect trust is earned via referrals, opinions, etc. On another hand, partial transitiveness implies that indirect trust often comes with special constraints, e.g. a maximum referral hop limit. It is desirable for a trust management scheme to take both direct and indirect trust into consideration, though it may assign different weights to them.

Trust is both *static* and *dynamic*. Static trust means that the value of trust does not change over time. Identity-based trust is typical static trust. Here the identity refers to information regarding an entity's social role and status as well as its relationship with other entities. For instances of identity-based trust, one can consider following two examples: Without any previous interactions, a private passenger car is reasonable to trust a police patrol car. Likewise, a truck trusts the leader of its fleet, which consists of the same type of trucks from the same company. Dynamic trust means that the value of trust changes over time. For example, interaction-based trust is dynamic.

Trust is *situation-dependent*. For example, an entity may adapt its threshold for a trusted decision, according to situation. In general, the threshold is supposed to be set higher in situations where a trust decision matters more, e.g. in safety applications. The threshold may also be adapted to the security level of the system [48]. For instance, an entity would have a lower threshold in a system with sophisticated cryptography than in one without cryptography.

2.2.1 Trust Metrics

While properties of trust often tell how trust can be measured, trust metrics tell what to be measured for evaluating trust. In ITS applications, a trust metric is often some character of an entity, e.g. honesty in message generation. Or it is some capability of an entity, e.g. the discernment capability for distinguishing between truthful and false messages, the capability for providing reliable timely and integral message delivery [71], etc. Here trust metrics are actually all entity trust

metrics. That is because for data we only consider the authenticity of it. Thus, there is no need for various metrics for data trust.

Trust metrics inherit the properties of trust. To understand this, we use an example trust metric to show how it can be utilized in trust management. Consider honesty in message generation as a specific trust metric. The value of this metric can be determined statically, i.e. initialized based on the role of an entity. E.g. a policeman possesses a higher static trust value in honesty than a regular driver. The value can also be dynamically changed over interactions. Besides, the value can be gained indirectly. Say entity *A* may get the value of trust in honesty of entity *B* from entity *C*. Finally, the trust threshold related to this metric may be adapted to situation as well. For example, suppose an entity decides to distrust any messages originated from entities with a trust value in honesty lower than a threshold. Such a threshold can be set higher in a dense network where information is abundant and thus the entity can be more selective in messages. In contrast the threshold should be lower in a sparse network where information is rare and the entity has to be more open to messages.

2.2.2 Potential Attacks

There are some common attacks [30] deliberately designed to sabotage trust management schemes. Those attacks include simple false information injection attacks, on-and-off attacks, Sybil attacks and collusion attacks. A simple false information injection attack happens when a malicious entity generates false information on purpose. An on-and-off attack happens when a malign entity behaves well or badly alternatively to dodge the detection. A Sybil attack [39] happens when a malign entity uses a large number of fake identities to beat the redundancy check of the network. For instance, an entity may ask opinions for a message from multiple different entities. A Sybil entity with many pseudonyms, i.e. bogus entities, can fake all those opinions using different pseudonyms and consequently fool the entity that asks for help. A collusion attack happens when

a group of well-coordinated malign entities contrive a conspiracy. We shall show how those attacks are handled when review related works of trust management in the next section.

Note that we consider traditional security problems like access controls, cryptography, etc. as a separate type of issues from the entity and data trust management problems investigated here. The difference between those two types of problems is first described in [92], which refers traditional security issues as *hard security* and trust management as *soft security*. Thus, traditional information security hazards such as modification of messages, denial of the generation of messages, etc. are not considered as attacks specially targeting trust management schemes. Those traditional hazards are typically prevented by employing asymmetric key cryptography. For example, [29, 53, 88, 111] use digital signatures to prevent malicious entities from modifying messages without detection. However it remains a separate important research direction to consider the traditional security issues in an ITS environment. For instance how to efficiently manage the public/private key pairs without revealing the privacy of entities in ITS environment needs to be studied. As an example, [94] suggests using anonymous key pairs to preserve sensitive information regarding the entity, such as owner, identity, routine, etc. [64] proposes an approach to deploy anonymous key pairs in VANETs.

2.3Survey on Trust Management for ITS

In this section, we provide our critical reviews of the existing works on trust management for ITS. Those works are organized based on the way in which the trust management is implemented.

For the purpose of validating a received message, both [38] and [29] consider a technique named *opinion piggybacking*. Opinion piggybacking means that each forwarding entity of a message appends their own opinion to the message and decides whether or not trust the message based on the attached opinions.

Specifically, in [38] each such opinion of a forwarding entity is a triple tuple including a continuous trust value o_{val} representing its trust value for the message, a discrete trust level $s \in \{1,2,3\}$ representing its trust value for the message's generator and its ID. The paper provides forwarding entities an algorithm calculating the values of their own o_{val} and s for a message by considering all the previous opinions attached with the message combined with locally stored trust values for the corresponding opinion providers. Dynamism is brought into the trust decision threshold by taking the spatial distance between locations of the message source entity and the deciding entity, as well as, the deciding entity's familiarity of the area into account. However, [38] provides little information regarding how trust values for entities get initialized and updated.

In contrast to [38], in [29] each opinion consists of a binary decision for the message, i.e. either trusted or not, a confidence value for the decision and the signature of opinion generator. Specifically, the paper adapts a clustering based routing protocol to propagate messages. The cluster-head entity calculates the trust value for the message by considering both the confidence of each attached opinion and its personal trust value for the opinion's corresponding generator. The cluster-head then makes a decision based on the calculation result to determine whether or not to relay the message. Consequently, only trusted messages get disseminated among different clusters.

An entity updates its trust values for other entities according to the following rules. Entity X's trust for entity Y is positively enforced if Y's opinion on a message leads to a correct decision; otherwise X's trust value for Y is reduced. The paper does not elaborate on how an entity gets to verify the correctness of a trust decision. An entity can absolutely discover the authenticity of the event reported in the message by direct observations. For example, an entity which has trusted a message reporting the clear of a road maintenance on its way to the destination would verify the decision is correct when it passes by the site. However, it is obvious that sometimes an entity has no way to witness an event reported by a once trusted message. For instance, an entity which has trusted a message reporting a jam on its future trajectory and thus decided to reroute, is unable to

verify the truth via direct observations. Thus, it is reasonable for an entity to confirm the authenticity of a reported event, if it receives a similar message from a highly trusted entity too. The threshold for the "high trustworthiness" can be customized by individual entities.

[29] proposes some methods for preventing on-and-off attacks. It suggests to defend such attacks by utilizing the "hard to win but easy to lose" principle. This principle imitates a practical norm in the real social life, namely, that trust value for an entity is difficult to build up but easy to tear down. Consequently, entities have to behave very discreetly all the time in order to keep their earned credits. A similar kind of attack, i.e. *betrayal attacks* where malicious entities first act normally to build up their trustworthiness but then abruptly start malign behaviors, can also be thwarted by following the same principle. Specifically, the scheme proposed in [5] employs a forgetting factor which allows trust values earned via interactions to decay over time. It also uses a larger value for the decrease factor than that for the increase factor. As a result, trust values for entities always gains slowly and loses fast over interactions.

One unique problem with the approach in [29] is regarding the confidence value of the opinion. Although the confidence value helps model the uncertainty of the opinion, there is no good guidance for entities to accurately estimate such a value. If unfortunately the estimation goes wild, the poorly calculated confidence value is going to damage the entity's trustworthiness badly in the feedback stage. In addition, the scheme is bonded with a cluster-based routing scheme. Therefore its applicability narrows dramatically.

Both schemes described in [29, 38] suffer from several other problems in the ITS environment. First is that a forwarding entity of a message is likely to have no previous interaction with the message's generator and have no ground to provide an opinion to the message. One solution is to ask the message generator embed its encrypted role into the message. Then forwarding entities can use static role-based trust value as the initial ground for opinion giving.

Second is that they only use a single trust value to measure entity trust. However, it is important for them to make a distinction between trust in honesty and trust in discernment. Trust in honesty measures the trustor's faith in the belief that the trustee will not produce falsified information whereas trust in discernment measures the trustor's faith in the trustee's ability of making correct trust decisions on messages. The difference between these two metrics reflects a similar situation in our social network: someone may be so honest that s/he never tells a lie however s/he may be also so inexperienced that s/he will be easily fooled by a lie. Likewise, entity *X*, with a high trust value in honesty but a low trust value in discernment implies that it barely generates false information but keeps making wrong decisions on messages due to reasons such as lack of interactions or surrounded by colluding entities. In such a case, another entity *Y* is expected to believe data originated from *X* but ignore *X*'s trust opinions for other messages. In this case, if two trust metrics are mixed and represented by a single value, *Y* is likely either over-estimate the credibility of *X*'s opinions or underestimate the reliability of data generated by *X*.

The last problem is most crucial. To show what the problem is, consider a scenario where a malicious entity *O* broadcasts false messages in one neighborhood for a while, flees to another far away area, say 5 miles from the old neighborhood, and starts to broadcast false messages again. It is likely that in the new area, there are few entities which have previous interactions with *O* and thus have no idea of *O*'s past bad behaviors. Consequently, the forwarding entities of falsified messages generated by *O* in the new area may make incorrect trust decisions in the beginning phase. Those entities may discover the evil of *O* after a period of time but by then *O* may have far gone again.

2.3.1 A Novel Trust Management Scheme

The last problem described above has its root in the fact that the proposed trust management schemes are fully decentralized and there is a lack of a central server which records the

trustworthiness of entities. This inspires us a novel idea of implementing trust management in a semi-centralized scheme.

Next we briefly describe how the scheme will work. The scheme assumes the existence of a central server, whose public key is globally known. Each entity registers with the server. The server stores reputation values of entities. Each entity can log in the server to file compliant or praise about another entity, of which the corresponding reputation value is updated by the server according to certain rules. Each entity is required to log in the server every T_{ru} time in order to download a certificate encrypted by the private key of the server. The certificate contains a statement associating the identity and the latest reputation value stored in the server of the entity. When an entity broadcasts a message, it is required to embed the certificate into the message. Consequently, any entity receives the message can decrypt the certificate using the server's public key and get a rough idea about the trustworthiness of the message's generator via the embed reputation. In addition, an entity is allowed to inquire the server about the reputation value of another entity any time if it suspects the reputation value in the received certificate is out of date. This means will efficiently prevent the aforementioned perpetrate-run-perpetrate type of attacks.

There are clearly some details need to be determined for the above scheme. For example, what is the best value for T_{ru} such that a good balance is stroke between the freshness of the reputation value and the efficiency of communication cost? Likewise, what are the rules of the server to update reputation value given the reported complain and praise? We leave those questions as future work and are going to implement this scheme in a forthcoming paper.

2.3.2 Opinion Inquiring

Akin to the opinion piggybacking approach, Minhas et al. [111] proposes to validate messages via considering other entities' opinions too. However, unlike in [29, 38] where opinions are bonded with the message forwarding protocol and consequently causes an entity to lose the option to choose opinion providers, the paper allows an entity actively to ask for opinions from entities picked by

itself. When an entity receives a message announcing an event, it asks opinions from *N* other most trusted entities based on the trust values output by the lower layer entity trust management, of which the details are briefly summarized below.

Similar to the approach in [29], the entity trust consists of role-based trust and interactions-based trust. A globally trusted certificate authority (CA) issues an encrypted certificate to each entity which binds the role and the public key of the entity. By requiring communicating parties to exchange certificates before interactions, role masquerading is efficiently prevented. Interaction-based trust is learned by an entity via past interactions. The principle of the learning process is the same as that in [29], namely "good" interactions get rewarded and "bad" interactions get punished. Each entity ranks the trustworthiness of other entities in a major order of role-based trust and a minor order of interaction-based trust.

Once having all the opinions for a received message, the entity uses an equation to calculate the trust value for the message. Within that calculation, each opinion is assigned with different weight in relation to factors including: (i), local role-based trust for the opinion provider; (ii), local interaction-based trust for the opinion provider; (iii), temporal closeness between the time when the event takes place and the time when the opinion is generated; (iv), spatial closeness between the location where the event takes place and the location where the opinion is generated. The output trust value is finally converted to a decision.

There is one unique problem with this opinion inquiring based scheme. Since the selection of entities for which opinions are asked is based on trust value rather than spatial closeness, the scheme may suffers a high communication cost and time delay, if unfortunately, the selected entities are far away.

2.3.3 Passive Majority Consensus

Patwardhan et al. [88] utilizes the fact that information is often redundant in ITS applications to validate a message. The paper assumes a network where anchored resources, such as parking

meters and roadside sensors, perpetually provide trustworthy data to surrounding entities. A message can be accepted, i.e. validated, by an entity via either using a majority consensus or directly communicating with the anchored resource which produces the message. A majority consensus at an entity *O* will validate a message *M* if (i) at the time of consensus, *O* has received at least *P* other messages which report the same event as message *M* does, all from different entities (ii) message *M* along with the other *P* messages consist of the majority opinion regarding the event, where *P* is a system parameter.

The proposed scheme takes a passive approach to wait for messages from other entities in contrast to the scheme in [111] which proactively asks for opinions from other entities. The drawback of the passive approach is that an entity may wait for a long time or even forever to receive enough messages of the same event required by the majority consensus in a sparse network. Thus, it is natural to consider a neutral approach combining the merits of the active and passive approaches. For example, adapt the state of activity of the entity to the network density and consider both trust value and geographical closeness as metrics when choosing entities for opinions. In addition, these two schemes also suffer from the same three problems of schemes in [29, 38] described before.

2.3.4Data Fusion Dependent

By far all discussed trust management schemes employ some variant of the majority scheme to integrate complementary evidences, i.e. opinions or messages of the same event for making a trust decision. In contrast, Raya et al. [95] introduces a trust management scheme emphasizing on exploiting different kinds of data fusion techniques for the purpose of complementary evidence integration. Specifically, a trust metric is used to measure an entity's vulnerability to attacks. The value of this metric is assigned statically. For instance, better equipped and closely monitored entities have higher static trust values than regular entities. Given all the available evidences of a particular event, the entity takes all the messages of the event and their corresponding weights to

output the trust value for the event by using certain data fusion technique. Similar to the approach in [111], the weight of each message is related to both the static trust value for the message's generator as well as time and distance closeness between the reported event and the message's generator. The data fusion techniques include voting, Bayesian inference and Dempster-Shafer Theory of evidence. Similar to the approach in [88], this scheme suffers in sparse network where sufficient evidence is not likely available for an event.

2.3.5 Position Verification

Golle et al. [53] also presents a data trust management scheme for VANETs without the use of any entity trust metric. The authors assume that each entity maintains a *model of the VANET* against which any incoming message will be validated. In its core, a model of the VANET is a set of observations about the VANET already known to an entity. All messages consistent with the entity's model of the VANET are validated; otherwise, the entity attempts to eliminate the inconsistency by ranking all the possible explanations and picking the simplest explanation.

The paper does not provide a general algorithm for validating data against the model of the VANET but uses examples to demonstrate the idea. For instance, the paper shows how the approach is used for prevent Sybil attacks. Specifically, drivers are supposed to be capable of broadcasting position statements pertaining to themselves and others. For example, driver A may broadcast a message stating "I am at location L_1 and I spot a police car at location L_2". The paper assumes that broadcasted position statements are immediately available to entities network wide. Now suppose in its model of the VANET, entity A has identified that neighbor B and C are indeed distinct entities. Further suppose that both B and C state that identity X and identity Y, which are far away from A, locate at the same position. However, at the same time X and Y themselves state they are at different positions. Clearly there is a conflict among statements received by A. There are two possible explanations which can resolve the conflict, namely (i) B and C collude to lie about X and Y (ii) X or Y is faked by a Sybil entity. A makes the choice by utilizing the so called *adversarial parsimony*

principle, which essentially picks the explanation involving fewest malicious entities. Thus, in this case, the second explanation, i.e. the potential Sybil attack is detected by entity *A*.

The problem of this approach is that it is not practical for each vehicle building and maintaining a model of the VANET in real time, since the paper assumes that any broadcasted statement made by an entity is instantly universally available to all other entities, i.e. ignoring the propagation time of messages. Besides, since Sybil attacks requires the lack of a central trusted authority such as a CA which provides identity authentication service, thus, the above approach is not necessary for ITS as entities may get certificates from trusted infrastructures, e.g. gas stations, parking lots or dedicated roadside facilities [114] to authenticate themselves. However, implementation level details for the approach such as which infrastructure are considered to be trusted, how certifications are managed, etc. are needed to be thought thoroughly.

2.3.6 Collusion Attacks Prevention

Few attempts have been done in preventing collusion attacks. Some existing trust management schemes for ITS are considered to have certain defense ability against collusion attacks of a particular form. For instance, the scheme proposed in [53] can prevent a special kind of collusion attack, i.e. position spoofing by a group of malicious entities. However, because of the nature of the adversarial parsimony principle, the scheme only works when even the simplest explanation includes a collusion attack. Trust management systems described in [29, 38] are also considered to be capable of absorbing colluded false message injection attacks to certain degree. Nevertheless the strength of such a capability is contingent upon factors like network density, complexity of the collusion, etc. In short, the principles of defending collusion attacks in general as well as concrete methods for preventing application-specific collusion attacks still wait to be further studied.

2.4 Discussion

Trust management is a crucial aspect for ITS applications, and yet remains an open problem. Existing works on trust management for ITS aim at different kinds of sub-problems. Unfortunately, there is a lack of classification and comparison of these works against a uniform backdrop. For this purpose, we present a survey on trust management for ITS. Specifically, our survey describes the properties of trust, trust metrics and potential attacks against trust management schemes. Existing related works are carefully reviewed and compared. We also contribute a novel idea of implementing trust management in a semi-centralized fashion. Our work is an important step toward building an efficient, comprehensive and reliable trust management scheme for ITS.

Chapter 3

Real-time Taxi-sharing with Smart Phones

3.1 Introduction

3.1.1 Background

Many transportation related problems, such as traffic congestions, difficult to find available parking spaces, hard to hail a taxi during rush hours, have been bringing inconvenience to average people's daily life for a long time. In the past, different methods have been mainly proposed to tackle these problems separately. In contrast, we aim to find a cure-all which solves or at least alleviates all these problems. One major reason accounts for the occurrence of these problems is that the ridership of vehicles is under-exploited as a resource. Thus, we study ridesharing as a promising means to improve the utilization of vehicle ridership and thus serve the purpose of fixing relevant transportation problems.

Ridesharing , at a high level, is defined as the practice that travelers share their partial or entire trips in a vehicle. Given this definition, we can immediately see three aspects that feature the characteristics of ridesharing, i.e. the type of vehicles used, the relationship among riders who share a trip, and the monetary relationship between riders and the driver of the vehicle. A *driver* is the person who drives the vehicle that provides the ride opportunities and a *rider* is a person who consumes a passenger seat of the vehicle.

Figure 4 further illustrates these aspects of ridesharing. Vehicles used for ridesharing can be cars and vans, and the corresponding ridesharing is conventionally referred to as *carpool* and *vanpool*, respectively. The relationship among riders who share a trip can be family members, acquaintances (e.g. colleagues), and strangers, and the corresponding ridesharing can be referred to as *Fampool*, *Co-worker Carpool*, *Casual Carpool* [28], respectively. Ridesharing can be

conducted for non-profit purposes (e.g. commute ridesharing) and for-profit purposes. For non-profit ridesharing, passengers may or may not pay the driver for the shared trips. Drivers may still have incentives to provide free rides even the passengers do not compensate them. These incentives often are provided by ridesharing-encouraging policies made by government authorities, such as the use of *High-Occupancy Vehicle (HOV)* lanes, toll reduction.

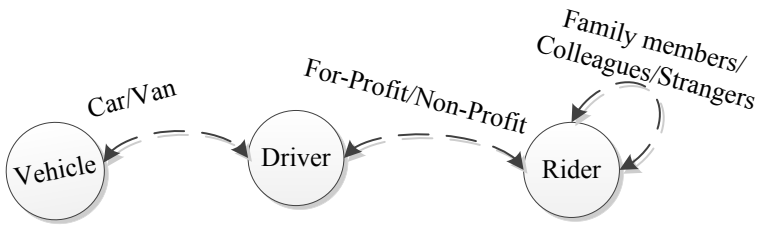


Figure 4 Elements of ridesharing

In other literature, ridesharing is also referred as to carpool, or car sharing. In this study, we distinguish these terms. Carpool is considered to be equivalent to non-profit ridesharing. Car sharing refers to the practice of mid-term or long-term renting or leasing cars. This word is popularized due to the emerging and fast growing of new car rental companies, notably Zipcar.

3.1.2 Motivation

Transportation problems, such as traffic jams, find parking slots, hard to hail a taxi during rush hours, are long-existing headaches for cities, especially those with a large population. These problems affect the environment, the economy, and more directly average people's daily lives by cost them a large chuck of time each day. For example, on average drivers spend 8.1 minutes in finding a curbside parking slot [102].

During the past, different methods have been mainly proposed to tackle these problems separately. For examples, extending the road network is one common approach to tackle traffic jams; sensors which detect the availability of parking spaces [4] are installed to help drivers find parking slots more quickly. However, those solutions often require additional construction or new equipment added to the existing infrastructures and thus are often costly. Also, their benefits are

usually limited to the specific corresponding problem. Though for some case, solution to one problem has positive effects on another, e.g. reducing searching time for parking slots help ease traffic jams [17].

In contrast, we aim to find a cure-all which is instrumental for solving all the problems at once. One major reason which accounts for the occurrence of the above transportation problems is that the passenger seats of vehicles is under-exploited as a resource. Thus, we study ridesharing as a promising means to improve the utilization of vehicle ridership and reduce the number of cars on the road. On one hand, ridesharing is a more cost-effective transport mode than private transport. On the other hand, it is also more flexible compared to public transport. First, ridesharing, e.g. taxi ridesharing, still has the potential to satisfy door-to-door travel needs of individuals; whereas public transport such as buses, trains typical only connect a few designated locations. Second, ridesharing is also more temporally flexible than public transport as the schedule of a shared trip can be the result of negotiation of participants instead of being fixed. For these reasons, ridesharing is an important transport mode complementary to the existing private and public transport, which is capable of satisfying travel needs while being cost-effective. We are going to model a generic ridesharing problem and study the benefit of ridesharing via both theoretical analysis and experiments.

3.1.3 Technical Challenge

Taxi is an important transportation mode between public and private transportations, delivering millions of passengers to different locations in urban areas. However, taxi demands are usually much higher than the number of taxis in peak hours of major cities, resulting in that many people spend a long time on roadsides before getting a taxi. Increasing the number of taxis seems an obvious solution. But it brings some negative effects, e.g., causing additional traffic on the road surface and more energy consumption, and decreasing taxi driver's income (considering that demands of taxis would be lower than number of taxis during off–peak hours).

Unfortunately, real-time taxi-sharing has not been well explored, though ridesharing based on private cars, often known as carpooling or recurring ridesharing, was studied for years to deal with people's routine commutes, e.g., from home to work [19, 25]. In contrast to existing ridesharing, real-time taxi-sharing is more challenging because both ride requests and positions of taxis are highly dynamic and difficult to predict. First, passengers are often lazy to plan a taxi trip in advance, and usually submit a ride request shortly before the departure. Second, a taxi constantly travels on roads, picking up and dropping off passengers. Its destination depends on that of passengers, while passengers could go anywhere in a city.

3.1.4 Contribution

We consider real-time taxi-sharing for a large number of taxis. We placed our problem in a practical setting by exploiting a real city road network and the enormous historical taxi trajectory data. The contribution of this section is summarized as follows.

We proposed and developed a taxi-sharing system using the mobile-cloud architecture. The Cloud integrates multiple important components including taxi indexing, searching, scheduling, and travel time estimation. Specifically, we propose a spatio-temporal indexing structure, a taxi searching algorithm, and a scheduling algorithm. Supported by the index, the two algorithms quickly serve a large number of real-time ride requests while reducing the travel distance of taxis compared with the case without taxi-sharing.

We built mobile clients running on smart phones, enabling taxi riders and taxi drivers to interact with the Cloud and with each other. Taxis are also mobile sensors constantly probing the traffic on road surfaces, and therefore providing a more accurate estimation of travel time for our system.

We performed extensive experiments to validate the effectiveness of taxi-sharing as well as the efficiency and scalability. According to the experimental results, the fraction of ride requests that get satisfied is significantly increased by 22% meanwhile riders save 5% in taxi fare via taxi-sharing

when the taxis are in high demand. Furthermore 2 million liter of gasoline can be saved each year in Beijing by taxis alone if taxi-sharing is allowed.

3.2 Related Works

3.2.1 Taxi Recommender and Dispatching Systems

Quite a few recommender systems have been proposed for improving an individual taxi driver's income and reducing unnecessary cruising. Based on historical taxi trajectories, Yuan and Zheng et al. [122, 124] proposed a system that suggests some parking places for an individual taxi driver towards which they can find passengers quickly and maximize the profit of the next trip. Similarly, Ge et. al [47] suggests a sequence of pickup points for a taxi driver. While these systems are only designed from the perspective of taxi drivers, our system considers the needs of both taxi drivers and riders.

Taxi dispatching services [73, 100, 120] usually send a taxi close to a passenger as per the passenger's call without considering taxi-sharing. Consequently, only vacant taxis need to be examined for each dispatch, which can be easily retrieved by answering a range query. In our case, each taxi that is occupied under full capacity needs to be considered. This complication introduces new challenges.

3.2.2 Dial A Ride Problem (DARP) and Its Applying Heuristics

The taxi ridesharing problem is relevant to the Dial-a-Ride Problem (DARP) [16], a.k.a. Vehicle Routing Problem with Time Windows [37] studied in the operation research community. DARP is essentially also a constraint satisfaction problem, i.e., planning schedules for vehicles, subject to the time constraints on pickup and delivery events. Static DARP can be viewed as the static single-hop non-profit ridesharing problem with additional presumptions (e.g. all vehicles are required to start trips and return after all trips from and to a depot location in DARP). Please see [33] for a more comprehensive survey on DARP.

DARP is NP-hard [101]. Therefore, majority efforts have been put into developing heuristics for the problem. Some heuristics such as genetic algorithms are also called meta-heuristics as these heuristics provide a general computation paradigm which is applied to solve a broad range of problems. For the sake of simplicity, we use heuristics to refer meta-heuristics as well. Figure 5 shows a hierarchy of common heuristics.

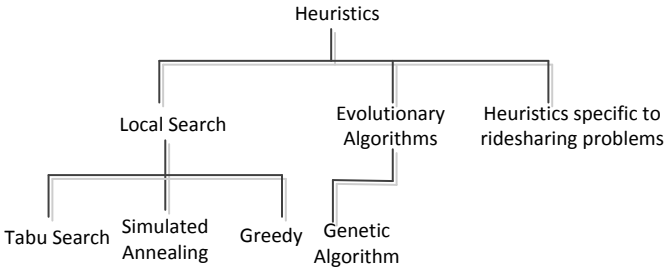


Figure 5 Hierarchy of meta-heuristics

Table 1 lists some heuristics that have been applied to static *Dial-A-Ride Problem (DARP)* in the existing literature.

Table 1 A list of heuristics applied to ridesharing problems

Reference	Meta-heuristics/heuristics	Formulation of Ridesharing Problem		
		Constraints	Optimization Objective	Max # of Queries
[26, 59]	Tabu Search	time window, vehicle capacity	travel distance	295
[21]	genetic algorithms	time window, vehicle capacity	linear combination of factors such as ride time, waiting time, time window violations.	144
[20]	simulated annealing	time window	objection function of weighted factors	N/A
[87]	variable neighborhood search	time window, vehicle capacity	a linear combination of violations	295
[27]	heuristics for graph assignment problem	time window, vehicle capacity	linear combination of # of served users and level of service (considering factors such as waiting time, riding time, etc.) of these users	180

[118]	inter-route exchanges, diversification	time window, heterogeneous vehicle capacity	linear combination of travel distance, travel time, waiting time.	2000
[62]	set cover heuristic	time window, vehicle capacity	a linear combination of travel time of riders and driver and travel distance	215

Among all heuristics, local search is the most frequent strategy used to solve ridesharing problems. In this section, we briefly describe some common local search heuristics. Figure 6 shows the logical flow and major components of a typical local search heuristic. We will walk through the flow chart by using the ridesharing problem as the context.

A *solution* P of a ridesharing problem is a set cover for the given set of queries S_Q. A *neighbor* of a set cover P, denoted by P', is obtained from P by an operation called *intergroup query move*. Each intergroup query move removes a query Q from a query group G_i and add it to another query group G_j (the cost of query group G_i and G_j should update accordingly).

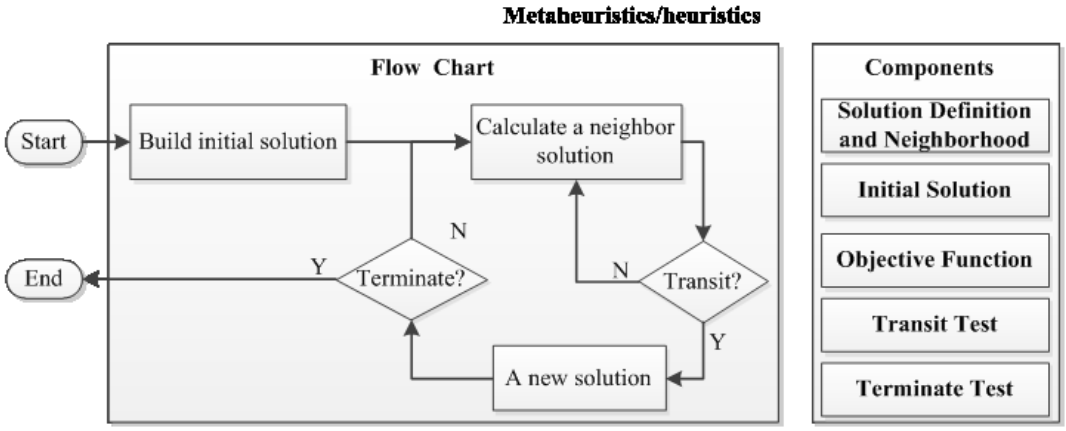


Figure 6 Flow chart and major components of a meta-heuristics

Existing works provide other definitions of the neighborhood of ridesharing solutions. For example, except the intergroup query move operation, [118] defines additional two operations that produce a neighbor: (i) remove two queries from one query group and add them to another group; (ii) two query groups exchange one of their queries. These two operations obviously can be

considered as a sequence of intergroup query move operation. However, it is not clear from the paper how these operations affect the performance of heuristics.

As shown in Figure 6, a local search heuristic works by starting with an initial solution and iteratively transiting to one of the neighboring solutions until the termination criterion is met. One way to get an initial solution [118] is to apply a greedy grouping algorithm. The algorithm iterates the query set and deals with each query Q in the following way: Q is added to the first query group such that the new group of queries obtained after addition can build a ridesharing schedule by themselves; if Q cannot join any group, then makes a new group by Q itself.

Transit test is the paramount component of a heuristic. The solution transits to one of its neighbor at each iteration. Denote by P_k and P_{k+1} the solution at iteration k and $k+1$ respectively. The transit test decides how P_{k+1} is chosen given P_k. If the transit test performs an exhaustive search, then all neighbors of P_k should be considered as the candidate of Pk+1; otherwise, only a part of neighbors of P_k need to be considered. In the latter case, the transit test typically chooses the first neighbor that meets certain requirement. As the specifics of transit test vary from heuristic to heuristic, we discuss some representative heuristics separately below. For the sake of description, denote by $P_k^{'}$ a neighbor of P_k and $\Delta = C(P_k^{'}) - C(P_k)$.

Greedy: If transit test is exhaustive, then P_{k+1} is the first neighboring solution $P_k^{'}$ leading to a lower value of the cost function, i.e. $\Delta < 0$. If no such solution exists, otherwise, P_{k+1} is the $P_k^{'}$ resulting the largest cost reduction among all neighbors of P_k.

Simulated Annealing: Comparing to the greedy heuristics, simulated annealing relaxes the acceptance condition by accepting the transition with certain probability even when the value of cost function increases. Specifically, if $\Delta < 0$, then the transition to $P_k^{'}$ is always accepted. If $\Delta \geq 0$, then the move to $P_k^{'}$ is accepted with probability $e^{-\frac{\Delta}{T}}$, where T is a parameter called the temperature, which changes during the course of the algorithm. Usually

T is large in the beginning and then it decreases after each iteration until it is close to 0. Different cooling schemes, i.e. ways in which the value of T is decreased as iteration goes, can be applied. Often T drops by multiplying a constant factor $a \in (0,1)$. The major disadvantage of simulated annealing is that the algorithm may get back to the solution already visited recently and get trapped there.

Tabu Search: Tabu search algorithm avoids frequently revisiting explored solutions by introducing a *tabu list*. A tabu list stores attributes of the previous few transitions. It has a fixed number of entries (usually between five and nine) and it is updated each time a new transition is accepted in a *First-In-First-Out (FIFO)* way: (i) the newly transition is added at the top of the tabu list to avoid returning to the same solution (to avoid returning to a local optimum); then (ii) all other entries in the list are pushed down one position; then (iii) the bottom entry is deleted. The rules of transit test in tabu search heuristics is described by Table 2.

Table 2 Rules for transit test in tabu search heuristics

	$\Delta < 0$	$\Delta \geq 0$
P_k' is not in the tabu list	the transition to P_k' is always accepted	The transition to P_k' is always rejected
P_k' is in the tabu list	the transition to $P_k'$is accepted only if $C(P_k')$ is smallest so far, i.e. smaller than that of any solution has been explored	a wait and see approach is applied: P_k remains as a candidate while the search continues for a neighbor which can be accepted immediately. If no such neighbor is found, transits to the best neighbor P_k'.

Terminate Test: local search based heuristics often terminate when the iteration number reach a pre-defined threshold.

3.2.3 Real-time Taxi-sharing

Though real-time taxi-sharing has been studied in several previous works [51, 86, 90, 107],

our work demonstrates three major advantages. First, our problem definition is more realistic by considering three different types of constraints. Some existing works (see [51, 107]) did not consider time window constraints and none of these previous works modelled monetary constraints. Second, we analyzed the computational cost of each component of the system, proposing a spatio-temporal index and a taxi searching algorithm, which significantly improve the system efficiency. Third, simulation results presented here is more convincing as we evaluated our system based on the real data and at a much larger scale than previous works did. Chen et al. [90] reported simulation results based on 1.5K synthesized ride requests, which is already the largest set among these works. In contrast, the size of the ride request stream in our experiment is as large as 20K and these ride requests are learned from the historical trajectory data set.

3.3 Problem Definition

The real-time taxi-sharing problem consists of a data model, constraints, and an objective function. We describe each part separately below before giving the formal definition of the problem.

3.3.1 Data Model

Ride Request: A *ride request* Q is associated with a timestamp $Q.t$ indicating when Q was submitted, a origin point $Q.o$, a destination point $Q.d$, a time window $Q.pw$ defining the time interval when the rider wants to be picked up at the origin point, and a time window $Q.dw$ defining the time interval when the rider wants to be dropped off at the destination point. The early and late bounds of the pickup window are denoted by $Q.pw.e$ and $Q.pw.l$ respectively. Likewise, $Q.dw.e$ and $Q.dw.l$ stand for that of the delivery window.

In practice, a rider only needs to explicitly indicate $Q.d$ and $Q.dw.l$, as most information of a ride request can be automatically obtained from a rider's mobile phone, e.g., $Q.o$ and $Q.t$. In

addition, we can assume that both $Q.pw.e$ and $Q.dw.e$ equals to$Q.t$, and $Q.pw.l$ can be easily obtained by adding a fixed value, e.g. 5 minutes, to $Q.pw.e$.

Taxi Status: A *taxi status* V represents an instantaneous state of a taxicab and is characterized by the following fields.

- $V.ID$: the unique identifier of the taxicab
- $V.t$: the time stamp associated with the status
- $V.l$: the geographical location of the cab at $V.t$
- $V.s$: the current schedule of V, which is a temporally-ordered sequence of origin and destination points of n ride requests Q_1, Q_2,…… Q_n such that for every ride request Q_i, i=1,…n, either 1) $Q_i.o$ precedes $Q_i.d$ in the sequence (referred to as the *precedence rule* thereafter), or 2) only $Q_i.d$ exists in the sequence.
- $V.r$: the current projected route of V, which is a sequence of road network nodes calculated based on $V.s$.

From the definition, it is clear that the schedule of a vehicle status is dynamic, i.e. changes over time. For example, a schedule involving 2 ride requests Q_1 and Q_2 could be $Q_1.o \rightarrow Q_2.o \rightarrow Q_1.d \rightarrow Q_2.d$ at a certain time. The schedule is updated to $Q_2.o \rightarrow Q_1.d \rightarrow Q_2.d$ once the taxi has passed point $Q_1.o$.

3.3.2 Constraints

The crux of the taxi-sharing problem is to dispatch taxis to ride requests, subject to certain constraints. We say that a taxi status V *satisfies* a ride request Q or Q is *satisfied* by V if the following constraints are met.

- *Vehicle Capacity Constraint*: the number of riders that sit in the taxicab does not exceed the number of seats of a taxi at any time.

- *Time Window Constraints*: all riders that are assigned to V should be able to depart from the origin point and arrive at the destination point during the corresponding pickup and delivery window, respectively.
- *Monetary Constraints*: these constraints provide certain monetary incentives for both taxi drivers and riders. That is, a rider does not pay more than without taxi-sharing; a taxi driver does not earn less than without taxi-sharing; when travelling the same distance; the fare of existing riders decreases when a new rider joins the trip.

3.3.3 Objective function and Problem Definition

Since multiple taxi statues may satisfy a ride request, an objective function is usually applied to find the optimal taxi. A variety of objective functions have been used in the existing literature, where a weighted cost function combining multiple factors such as travel distance increment, travel time increment and passenger waiting time, is the most common [21, 26, 117, 118]. In this study, given a ride request, we aim to find the taxi status which satisfies the ride request with minimum increase in travel distance, formally defined as follows: *given a fixed number of taxis traveling on a road network and a sequence of ride requests in ascending order of their birth time, we aim to serve each ride request Q in the stream by dispatching the taxi which satisfies Q with minimum increase in travel distance on the road network*.

This is obviously a greedy strategy and it does not guarantee that the total travel distance of all taxis for all ride requests is minimized. However, we still opt for this definition due to two major reasons. First, the real-time taxi-sharing problem inherently resembles a greedy problem. In practice, taxi riders usually expect that their requests can be served shortly after the submission. Given the rigid real-time context, the taxi-sharing system only has information of currently available ride requests and thus can hardly make optimized schedules based on a global scope, i.e. over a long time span. Second, the problem of minimizing the total travel distance of all taxis for the complete ride request stream is NP-complete. We prove this statement as follows. The problem

of optimizing travel distance for all taxis for the whole query stream, denoted by *Total Distance Optimization Taxi Ridesharing Problem (TDOTRP)*, can be formalized as the following decision problem: given a stream of queries S_Q, a start time t_s (t_s is the smallest value among the birth time of any query in S_Q) and a set of taxi statuses S_V at t_s, a road network RN in which each road segment is associated with a speed limit, a number $P \in [0,100]$ and a number $D \geq 0$, plan a schedule for each taxi such that the total travel distance of all taxis is no larger than D and the fraction of satisfied queries is at least P precent. The TDOTRP is NP-complete because we can prove that it is a generalization of the *Travelling Salesman Problem with Time Window (TSPTW)*, which has already been proved to be NP-complete. The input of a TSPTW instance includes a start time t_0, N vertices $\{1, 2,..., n\}$ in which vertex 1 is the depot vertex, the pair-wise distances between vertices and a number $D' \geq 0$. Each vertex i is also associated with a time window $i.w = < e_i, l_i >$, where $l_i \geq e_i \geq t_0$ for all $i = 1, \ldots, n$. The question is to find out whether or not there is a cycle route of distance no larger than D' such that a salesman can leave the depot, i.e. vertex 1 at t_0, visit each vertex i ($i = 1, 2, \ldots, n$) once within their corresponding time window and return to the depot.

An instance of the TDOTRP I_{TDOTRP} can be constructed from an instance of the TSPTW problem I_{TSPTW} by: (i) create the road network of I_{TDOTRP} using the vertex pair-wise distance of I_{TSPTW}; (ii) place one vacant taxi at vertex 1 and let the start time $t_s = t_0$; (iii) create a query Q_i for each vertex i such that $Q_i.o = Q_i.d = i$, and $Q_i.wp = Q_i.wd = i.w$, $Q_i.t = t_0$ for $i = 1, \ldots, n$; In other words, every vertex i ($i = 1, \ldots, n$) of I_{TSPTW} is considered as a dummy query of which the pickup point (time window) coincides the delivery point (time window) and the query is known a priori; (iv) let P=100, which means I_{TDOTRP} needs to satisfy all the queries, and $D = D'$.

The above construction completes the proof that TDOTRP is a generalization of TSPTW. Since TDOTRP is clearly in NP, therefore, we have proved that TDOTRP is NP-complete.□

3.4 System Architecture

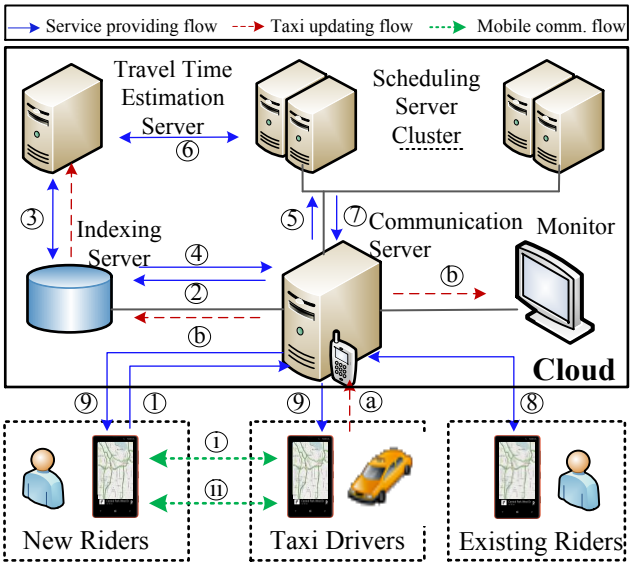


Figure 7 The architecture of the real-time taxi-sharing system

The architecture of our taxi-sharing system is presented in Figure 7. The Cloud consists of multiple servers for different purposes and a monitor for administers to oversee the running of the system (denoted as the red broken arrow (d)). Taxi drivers and riders use the same smart phone App to interact with the system, but provided with different user interfaces by choosing different roles, as shown in Figure 8 (a).

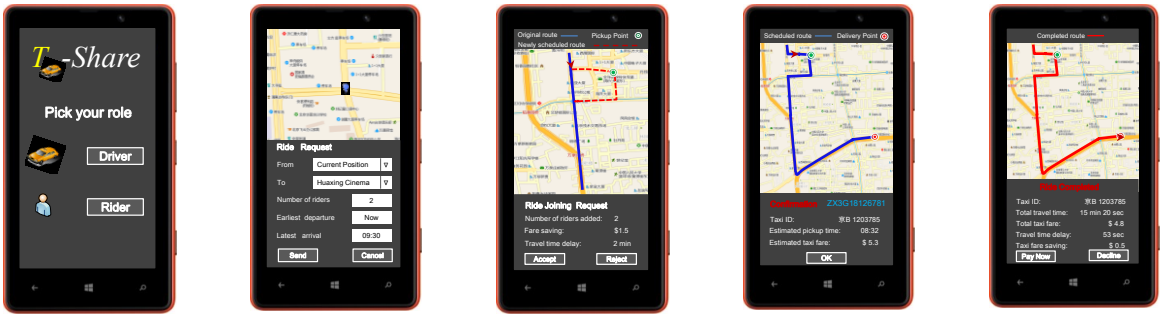
As shown by the red broken arrow (a), a taxi automatically reports its location to the Cloud via the mobile App when (i) the taxi establishes the connection with the system, or (ii) a rider gets on and off a taxi, or (iii) at a frequency (e.g., every 20 seconds) while a taxi is connected to the system. We partition a city into disjoint cells and maintain a dynamic spatio-temporal index between taxis and cells in the indexing server, depicted as the broken arrow (b). Additionally, we employ the T-Drive technique (see [121, 122]) using the GPS data of taxis stored in the indexing server to estimate the travel time of a route, denoted as the broken arrow (c).

Denoted by the solid blue arrow ①, a rider submits a new ride request Q to the *Communication Server*. Figure 8 (b) shows the corresponding interface on a rider's smart phone where the blue pin

stands for the current location of the rider. All incoming ride requests of the system are streamed into a queue and then processed according to the first-come-first-serve principle. For each ride request Q, the communication server sends it to the *Indexing Server* to search for candidate taxis S_V that are likely to satisfy Q, depicted as the blue arrow ②. Using the maintained spatio-temporal index and the travel time of routes from the *Travel Time Estimation*

Server (as shown by the blue arrow ③), the indexing server returns S_V to the communication server, denoted by the blue arrow ④.

Represented by the blue arrow ⑤, the communication server sends ride request Q and the received candidate taxi set S_V to the *Scheduling Server Cluster*. The scheduling cluster checks whether each taxi in S_V can satisfy Q in parallel, and returns the qualified taxi status that results in minimum increase in travel distance and a detailed schedule, shown as arrow ⑦. The travel time estimation server also needs to be accessed in this process (arrow ⑥).



(a) Pick a role (b) A new request (c) Ride request notification (d) Ride request confirmation (e) Ride Completed

Figure 8 Screenshots of the mobile client for riders

Each rider R who has been already assigned to the taxi will be enquired whether they would like to accept the join of the new rider, as depicted by blue arrow ⑧. The information, such as the estimated fare saving and travel time delay due to Q's join, will be displayed on their smart phones shown in Figure 8 (c). Rider R accepts the route change if she thinks the fare saving is worth the travel time delay. Otherwise, she can veto the route change by clicking the "Reject" button. The system remembers the rider's choice, automatically rejecting a route change in future if the ratio of

the fare saving to the travel time delay is smaller than the largest value the rider has ever rejected. Thus, a taxi passenger will not be bothered often.

After all the rider R accepted the route change, the new rider of Q gets a confirmation on her smart phone, as illustrated in Figure 8 (d). The confirmation informs the new rider the taxi ID, estimated pickup time and fare, the scheduled route, and a unique reservation code. The new schedule and the same reservation code are sent to the driver's phone at the same time. The reservation code will be used to build a connection between the phones of the new rider and the taxi driver when the new rider gets on the taxi. On the driver side, the smart phone displays a taxi's schedule, e.g., the next pickup and delivery points as well as the route, as illustrated in Figure 9.

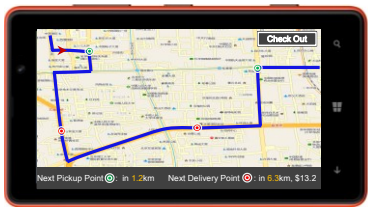
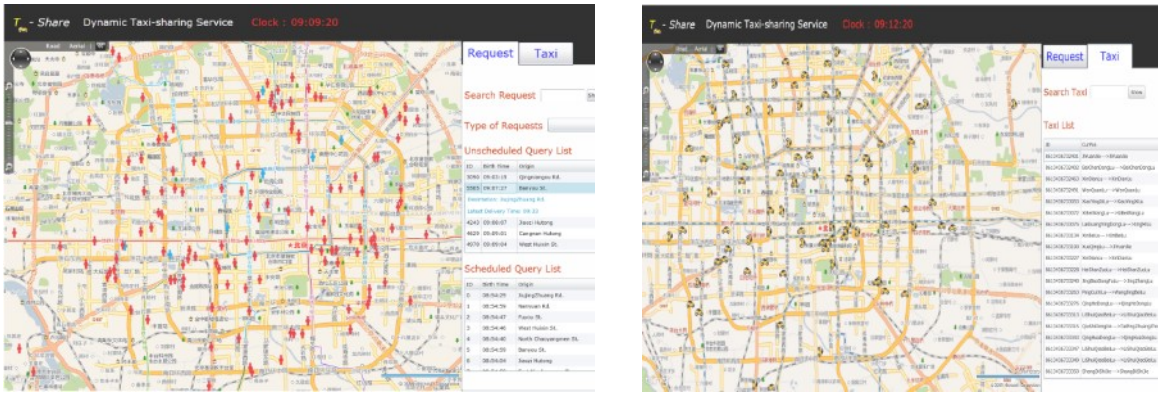


Figure 9 Screenshot of the mobile client for the driver

When a rider's trip is completed, the rider's APP will show the exact information, such as the actual fare and travel time, as illustrated in Figure 8 (e). The reservation code will be used again to confirm the payment to the Cloud. Interactions among riders, drivers, and the Cloud during pickup and drop-off events are detailed later.

The system administrator oversees the taxi-sharing system via the monitor. The monitor provides two views: one for ride requests, the other for taxis. Figure 10 (a) shows a screenshot of the ride request view, where all requests are displayed on the map at their corresponding pickup point, with scheduled requests in red and unscheduled requests in blue. On the right, two boxes list the detail information of scheduled and unscheduled ride requests respectively. The search box allows the administrator to quickly locate a request on the map via a request ID. Figure 10 (b) shows a screenshot of the taxi view of the monitor. Each taxi is represented by a yellow taxi symbol on the map. The locations of these symbols on the map are updated while the corresponding taxis upload

new statuses. Similarly, the search box is used to quickly locate and track a taxi via querying a specific taxi ID.



(a) Screenshot of the ride request view of the monitor

(b) Screenshot of the taxi view of the monitor

Figure 10 Screenshots of the monitor

3.5 Taxi Searching

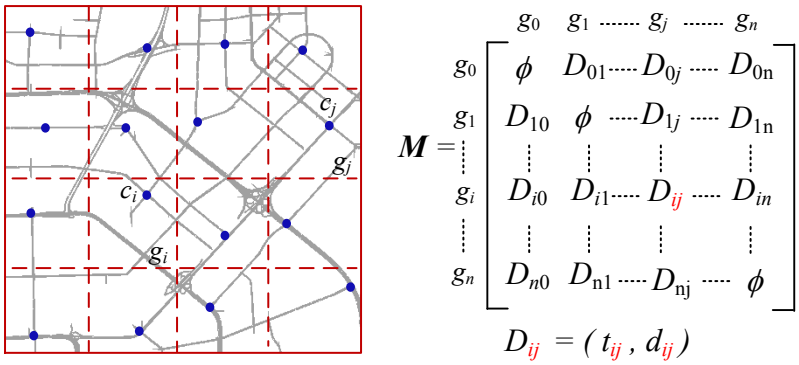
The taxi searching module quickly selects a small set of candidate taxis with the help of the spatio-temporal index. In this section, we will first describe the index structure and then detail the searching algorithm.

3.5.1 Index of Taxis

The spatio-temporal index of taxis is built for speeding up the taxi searching process. Specifically, we partition the road network using a grid. (Other spatial indices such as *R* tree can be applied as well, but we envision that the high dynamics of taxis will cause prohibitive cost for maintaining such an index.) As shown in Figure 11 (a), within each grid cell, we choose the road network node which is closest to the geographical centre of the grid cell as the *anchor node* of the cell (represented by a blue dot in Figure 11 (a). The anchor node of a grid cell g_i is thereafter denoted by c_i. We compute the distance, denoted by d_{ij} , and travel time, denoted by t_{ij}, of the quickest path on the road network for each anchor node pair c_i and c_j. The distance is only computed once while the travel time is calculated by the travel time estimation server once in a while (e.g. every 10 minutes).

Intuitively, we can use the computed travel time to quickly filter out a large number of taxis whose schedule is "far away" from a given ride request. The distance and travel time results are saved in a matrix as shown in Figure 11 (b). The matrix is thereafter referred to as the *grid distance matrix*.

Supposing all the nodes of the road network in a cell fall to its anchor node, the distance between any two arbitrary nodes equals to the distance between two corresponding anchor nodes. In other words, the grid distance matrix provides an approximation of the distance between any two nodes of the road network. These approximated distances avoid the expensive computation cost of frequent quickest path calculations at the stage of taxi searching.



(a) Grid-partitioned map (b) Grid distance matrix

Figure 11 Grid partitioned map and grid distance matrix

As illustrated in Figure 12, each grid cell g_i maintains three lists: a *temporally-ordered grid cell list* ($g_i.l_g^t$), a *spatially-order grid cell list* ($g_i.l_g^s$), and a *taxi list* ($g_i.l_v$). $g_i.l_g^t$ is a list of other grid cells sorted in ascending order of the travel time from these grid cells to g_i (temporal closeness). Likewise, $g_i.l_g^d$ is a list of other grid cells sorted in ascending order of the travel distance to g_i (spatial closeness). The spatial and temporal closeness between each pair of grid cells are measured by the values saved in the grid distance matrix shown in Figure 11 (b). For example, t_{2i} measures the temporal closeness from g_2 to g_i, and d_{2i} measures the spatial closeness from g_2 to g_i. The spatial grid cell list is only computed once. The temporal grid cell list is computed each time when travel times t_{ij}'s are updated. It is worth mentioning that cells that are neighbours in the grid may

not be the neighbours in a grid cell list because the distance is measured in the road network instead of a free space.

The taxi list $g_i.l_v$ of grid cell g_i records the IDs of all taxis which are scheduled to enter g_i in near future (typically within a temporal scope of one or two hours). Each taxi ID is also tagged with a timestamp t_a indicating when the taxi will enter the grid cell. All taxis in the taxi list are sorted in ascending order of the associated timestamp t_a. $g_i.l_v$ is updated dynamically. Specifically, taxi V_j is removed from the list when V_j leaves g_i; taxi V_k is inserted into the list when V_k is newly scheduled to enter g_i. If taxis are tracked (see [115]), when new GPS records are received from taxis, taxi lists need to be updated. Specifically, when a new GPS record from V_p is received, denote by g_q the current cell in which V_p is located, the timestamp associated with V_p in the taxi list of cell g_q and cells to be passed by V_p after g_q need to be updated.

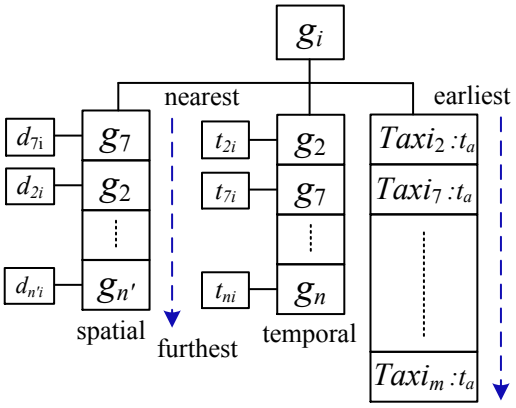


Figure 12 Spatio-temporal index of taxis

3.5.2 Taxi Searching Algorithms

The proposed algorithm is a bi-directional searching process which selects grid cells and taxis from the origin side and the destination side of a ride request simultaneously.

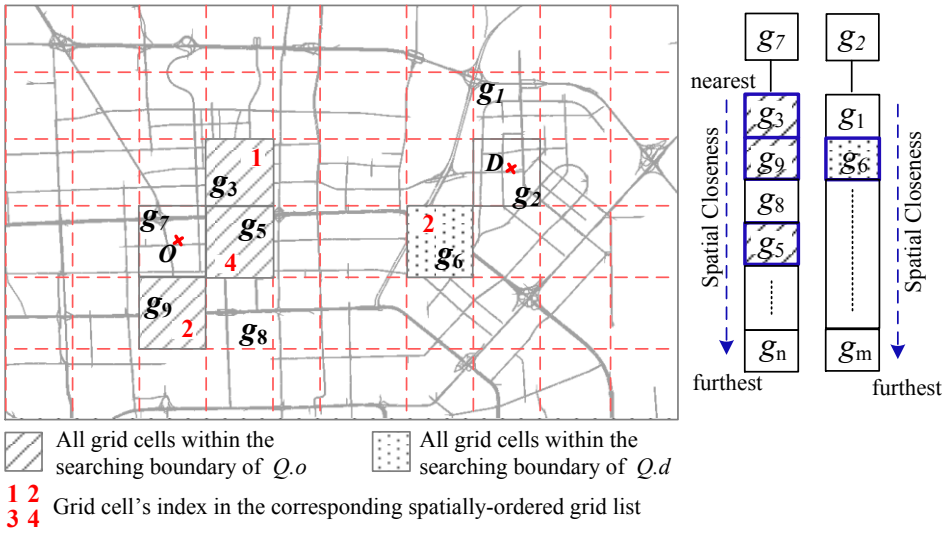


Figure 13 Overview of the dual-side taxi searching algorithm

To dive into the details of the algorithm, consider the ride request illustrated in Figure 13 where g_7 and g_2 are the grid cells in which $Q.o$ and $Q.d$ are located respectively. Squares filled with stripes stand for all possible cells searched by the algorithm at $Q.o$ side. These cells are determined by scanning $g_7.l_g^t$, the temporally-order grid cell list of g_7. That is, each grid cell in $g_7.l_g^t$ which holds Eq. (3.1) is a candidate cell to be searched at the origin side. Eq. (3.1) indicates that any taxi currently within grid cell g_i can enter g_7 before the late bound of the pickup window using the latest travel time between the two grid cells (assuming each grid cell collapses to its anchor node). The red number in each such grid cell indicates its relative position in $g_7.l_g^s$, the spatially-ordered grid list of g_7.

$$t_{cur} + t_{i7} \leq \ Q.pw.l \tag{3.1}$$

Squares filled with dots indicate the candidate grid cells to be accessed by the searching algorithm at $Q.d$ side. Likewise, each such grid cell g_j is found by scanning $g_2.l_g^t$ to select all grid cells which holds Eq. (3.2), which indicates that any taxi currently in g_j can enter the g_2 before the late bound of the delivery window (assuming that each grid cell collapses to its anchor node). In this example, g_6 is the only satisfying grid cell as shown by Figure 13.

$$t_{cur} + t_{j2} \leq Q.dw.l \tag{3.2}$$

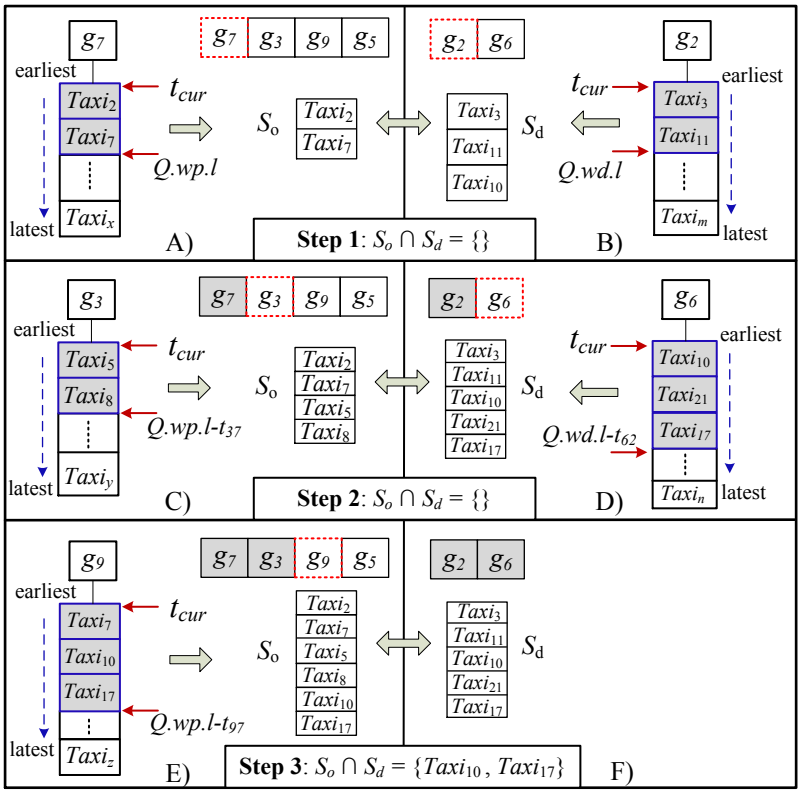


Figure 14 Calculation of the taxi set in the taxi searching process

Figure 14 then illustrates the searching process step by step. The algorithm maintains a set S_o and a set S_d to store the taxis selected from $Q.o$ side and $Q.d$ side respectively. Initially, both S_o and S_d are empty. The first step in the searching is to add the taxis selected from taxi list $g_7.l_v$ to taxi set S_o as depicted in Figure 14 (a), and add the taxis selected from taxi list $g_2.l_v$ to taxi set S_d as depicted by Figure 14 (b). Then the algorithm calculates the intersection of S_o and S_d. If the intersection is not empty, the algorithm stops immediately and returns the intersection set. Otherwise, it expands the searching area by including one other grid cell at each side at a time.

To select next cells, we use the following heuristic: for a taxi V, the closer one cell to be passed by V is to g_7 and the closer one cell to be passed by V is to g_2 (measured in the distance between the anchor nodes of the cells), the smaller V's scheduled travel distance increases after the insertion of the ride request. For the purpose of minimizing increased travel distance, the next grid cell

included at $Q.o$ side is always chosen as the next element in the spatially-ordered grid list $g_7.l_g^s$ which holds Eq. (3.1). Similarly, the next grid cell included at $Q.d$ side is always chosen as the next element in the spatially-ordered grid list $g_2.l_g^s$ which holds Eq. (3.2).

In this example, since S_o and S_d produces an empty intersection, the algorithm expands at $Q.o$ side to include g_3 (indicated by the broken red rectangle) and add taxis selected from $g_3.l_v$ as depicted in Figure 14 (c). At $Q.d$ side, the algorithm covers g_6 and adds taxis as indicated in Figure 14 (d). Unfortunately, the intersection set of S_o and S_d remains empty. Consequently, the algorithm needs to continue expanding the searching area at both sides. g_9 is then selected at $Q.o$ side; but no grid cell can be further included at the $Q.d$ side. After adding the taxis selected from $g_9.l_v$ into set S_o as shown in Figure 14 (e), we find $Taxi_{10}$ and $Taxi_{17}$ as the intersection between S_o and S_d. Hence, the searching algorithm terminates.

It is worth mentioning that an alternative approach is to search taxis solely from the origin side, that is, only consider taxis currently "near" the origin point of a ride request. The disadvantage of this alternative approach is that the number of selected grid cells could be large and thus it results in many taxis retrieved for the later scheduling process. In other words, it increases the overall computation cost, which is certainly not desirable for a rigid real-time system like taxi-sharing. Though the bi-directional searching algorithm may result larger increase in travel distance for the given ride request, as a compensation for the small loss in distance optimality, the algorithm selects far fewer taxis for the schedule allocation step, reducing the computation cost and ride request processing time. We found in the experiments that the number of selected taxis is reduced by 50% while the increase in travel distance is just 1% over the single-side search algorithm.

3.6 Taxi Scheduling

Given the set of taxi statuses S_V retrieved for a ride request Q by the taxi searching algorithm, the purpose of the taxi scheduling process is to find the taxi status in S_v which satisfies Q with minimum travel distance increase.

To this end, given a taxi status, theoretically we need to try all possible ways of inserting Q into the schedule of the taxi status in order to choose the insertion which results in minimum increase in travel distance. All possible ways of insertion can be created via three steps: (i) reorder the points in the current schedule, subject to the precedence rule, i.e. any origin point precedes the corresponding destination point (we refer to this step as the *schedule reordering* thereafter); (ii) insert $Q.o$ into the schedule (iii) insert the $Q.d$ into the schedule. The capacity and time window constraints are checked in all three steps, during which the insertion fails immediately if any constraint is violated. The monetary constraints are then checked after all three steps have been done successfully.

Consider a schedule with n points, among which m points are origins. After the schedule reordering step, there will be as many as $\frac{n!}{2^m}$ sequences which comply with the precedence rule. Though reordering the schedule is theoretically necessary for finding the optimal insertion way, we find that it is not the case in practice via experiments. For the sake of simplicity, in the rest of this section, the schedule reordering step is not considered unless otherwise stated.

Next we describe how to check the feasibility of each insertion possibility, subject to the capacity and time window constraints first and then the monetary constraints, given a pair of Q and V.

3.6.1 Time Window Constraints

Given a schedule of n points, there is clearly $O(n^2)$ ways to insert a new ride request into the schedule. For example, Figure 15 shows one way of inserting a request into a schedule with four points. To insert $Q_3.o$ after point $Q_1.o$ optimally, the algorithm needs to find the first path (starting from the shortest path) from $Q_1.o$ to $Q_3.o$ which allows the taxi to arrive at $Q_3.o$ during $Q_3.pw$ given the scheduled arrival time at $Q_1.o$. Since the shortest path is often not the quickest one when considering real road traffic, it is likely that multiple paths needs to be calculated before finding the first satisfactory path from $Q_1.o$ to $Q_3.o$. Similar process is required for other connecting paths, as illustrated by the dash lines in Figure 15. As a result, the overall computation load can be

extremely high for checking just one insertion way. To ease the computation load, here we only consider using the quickest path from one point to another during the insertion, though the new route may not be the shortest one in theory.

Denote by → the travel time of the latest quickest path from one location to another location calculated by the travel time estimation server in the Cloud, and t_w represents the time spent waiting for the passenger if the taxi arrives $Q_3.o$ ahead of $Q_3.pw.e$. Eq. (3.3) gives the travel time delay, denoted by t_d after inserting $Q_3.o$ between $Q_1.o$ and $Q_2.o$.

$$t_d = (Q_1.o \rightarrow Q_3.o) + (Q_3.o \rightarrow Q_2.o) + t_w - (Q_2.o \rightarrow Q_1.o) \tag{3.3}$$

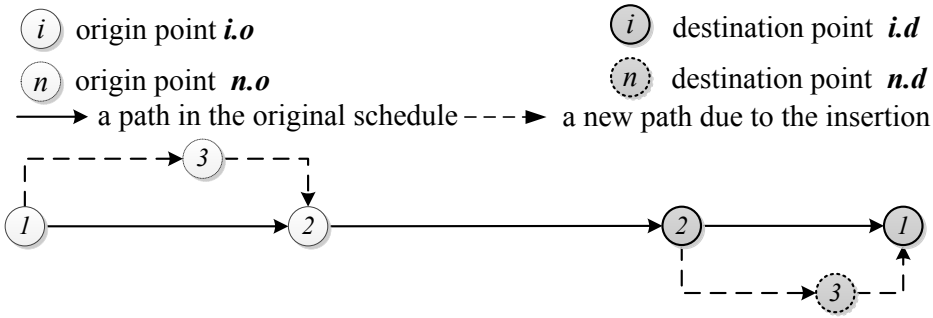


Figure 15 One possible insertion of a ride request into a schedule

If t_d results the late arrival at any point after $Q_2.o$ in the original schedule, then the insertion fails. For this purpose, we introduce the notion of slack time. Denote by a_p and a_d the projected arrival time at a pickup point $Q.o$ and a delivery point $Q.d$, respectively. Then the slack time at $Q.o$ and $Q.d$, denoted by $(Q.o)_{.st}$ and $(Q.d)_{st}$ respectively, is calculated by Eq. (3.4) and Eq. (3.5), respectively.

$$(Q.o)_{st} = Q.pw.l - a_p \tag{3.4}$$

$$(Q.d)_{st} = Q.dw.l - a_d \tag{3.5}$$

Thus, we can use slack times as a shortcut to check whether the delay incurred due to an insertion destroys the timely arrivals at any subsequent point in the schedule. In the example shown by Figure 15, if $t_d \geq Min\{(Q_1.d)_{st}, (Q_2.d)_{st}\}$, then the insertion fails. If $Q_3.o$ is inserted

successfully, the system proceeds to insert $Q_3.d$ in a similar way. Algorithm 1 summaries the process of computing a new route after the insertion of a new ride request.

Algorithm 1: Computing the new schedule and route after an insertion

Data: Ride request Q, taxi status V, insertion position i for $Q.o$, insertion position j for $Q.d$, current time t_{cur}
Result: Return *new_schedule* if the insertion succeeds; otherwise return False.

```
if t_cur + (V.l → Q.o) > Q.pw.l then /* cannot arrive Q.o on time */
    return False
if the time delay incurred by the insertion of Q.o causes the slack time of
any point after position i in schedule s smaller than 0 then
    return False
new_schedule ⟵ insert Q.o into V.s at position i   /* the slack time
of each pickup(delivery) point after position i is updated
accordingly at the meantime */
t_j ⟵ the scheduled arrival time of the j^th point of V.s
l_j ⟵ the geographical location of the j^th point of V.s
if t_j + (l_j → Q.d) > Q.dw.l then   /* cannot arrive Q.d on time */
    return False
if the time delay incurred by the insertion of Q.d causes the slack time of
any point after position j in new_schedule smaller than 0 then
    return False
new_schedule ⟵ insert Q.d into new_schedule at position j      /* the
slack time of each pickup(delivery) point in new_schedule is
updated accordingly at the meantime */
return new_schedule
```

3.6.2 Monetary Constraints

The new schedule after the insertion, by far, has only been checked against the capacity and time window constraints. It should also meet the monetary constraints. In this section we formulate the monetary constraints of taxi-sharing.

On one hand, we impose two constraints which encourage riders to participate in taxi-sharing by rewarding them with certain monetary gains. The *first rider monetary constraint* says that any rider who participates in taxi-sharing should pay no more than what she would pay if she takes a taxi by herself. The *second rider monetary constraint* says that if a occupied taxi V is to pick up a new rider Q, then each rider P whose travel time is lengthen due to the pickup of Q, should get a taxi fare cut; and the fare cut should be proportional to P's travel time delay.

On the other hand, we enforce one constraint which gives the driver stimulation to participate in taxi-sharing. This constraint says that a driver should earn for all distances she has travelled.

Intuitively the driver should make profit even for the distance of the reroute incurred by the join of a new passenger.

Now let us consider these three monetary constraints together in the scheduling context: given a taxi status V and a new ride request Q_n, under what conditions V will not violate the above three monetary constraints regarding to Q_n.

Denote by $Q_1,\ldots Q_{n-1}$ the riders involved in the current schedule of V before the join of Q_n. Also denote by d_i the distance between $Q_i.o$ and $Q_i.d$ on the road network , $i = 1, \ldots, n$. Denote by f_i the taxi fare of rider Q_i if V picks up Q_n. Denote by $F: R^+ \to R^+$ the *fare calculation function*, which maps the travelled distance to the taxi fare and is defined by some transportation authority or taxi company. Then the first rider monetary constraint can be expressed by Eq. (3.6).

$$f_i \leq F(d_i), i = 1, \ldots, n \tag{3.6}$$

Denote by M the revenue of the driver if she picks up Q_n and by D the distance of the new route after the pickup. Then the driver monetary constraint is expressed by Eq. (3.7).

$$M \geq F(D) \tag{3.7}$$

Since $M = \Sigma f_i$, we then have Eq. (3.8) by bridging two equations above.

$$F(D) \leq M = \Sigma f_i \leq \Sigma F(d_i), i = 1, \ldots, n \tag{3.8}$$

M can take any value between $F(D)$ and $\Sigma F(d_i)$ to make Eq. (3.8) stand. Here we take the lower bound $F(D)$ in order to reduce the total taxi fare of riders. Therefore, we have $M = F(D)$.

Then we need to distribute the total fare M to each individual rider. Denote by Δf_i the decrease in taxi fare for rider Q_i after the join of Q_n, $i = 1, \ldots, n-1$ and ΔT_i the travel time delay of rider Q_i's trip due to the pickup. The fare is determined in the way expressed by Eq. (3.9) and Eq. (3.10), where ΔD is the travel distance increase of the taxi route due to the join of Q_n and $f \geq 0$ is a constant.

$$f_n = F(d_n) - f \tag{3.9}$$

$$\Delta f_i = \frac{\Delta T_i}{\sum_{i=1}^{n-1} T_i}[(F(d_n) - f) - F(\Delta D)],\ i = 1, \ldots, n\text{-}1 \tag{3.10}$$

Eq. (3.9) says that the new rider can pays less by f than whatever she would pay if she rides alone. Eq. (3.10) says that (i) existing riders collectively save an amount which equals to the difference between the charge of the new rider and the driver's expected fare increase due to the increase in travel distance; and (ii) existing riders split the total saving proportional to their individual travel time delay (the second rider monetary constraint). Since it requires $\Delta f_i \geq 0$, therefore, we have Eq. (3.11).

$$F(d_n) \geq F(\Delta D) + f \tag{3.11}$$

Eq. (3.11) by itself is the sufficient and necessary condition for that taxi V does not violate any monetary constraint with respect to Q_n.

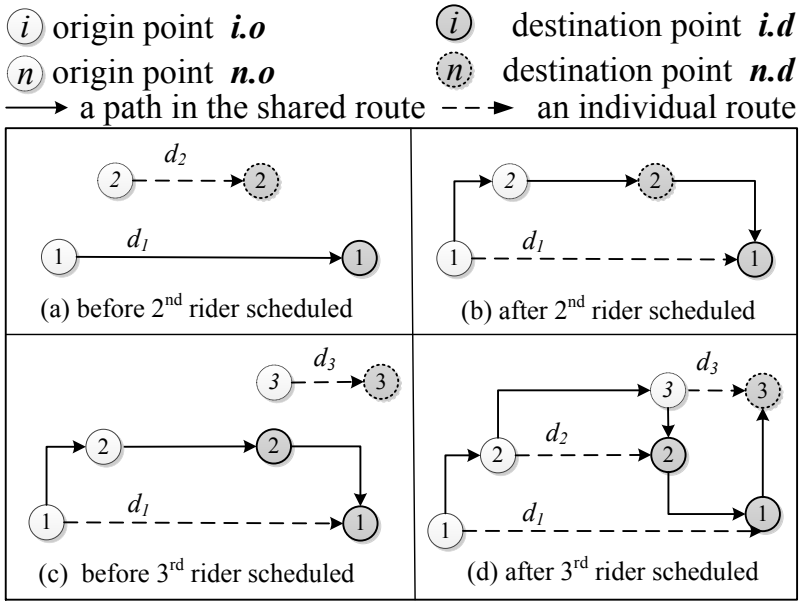


Figure 16 An example of the pricing constraint

Figure 16 illustrates how to apply the monetary constraints with a concrete example. Figure 16 (a) shows the schedule of a taxi before the second rider boards. The fare of the first rider is $f_1 = F(d_1)$. The monetary constraint for picking up the second rider is $F(d_2) \geq F(\Delta D) + f$, where ΔD

is the increase in travel distance due to the join of the second rider. If the above constraint stands, then we have $\Delta f_1 = [F(d_2) - f] - F(\Delta D)$ and $f_2 = F(d_2) - f$. Likewise, Figure 16 (c) shows the schedule of the taxi after the second rider joins and before the third rider joins. Similarly, the pricing constraint for picking up the third rider is $F(d_3) \geq F(\Delta D') + f$, where $\Delta D'$ is the increase in travel distance due to the join of the third rider. If this constraint stands, then we have $\Delta f_i = \frac{\Delta T_i}{\Delta T_1 + \Delta T_2}[F(d_3) - f] - F(\Delta D')), i = 1, 2$ and $f_3 = F(d_3) - f$.

In practice, some rider may think the taxi fare decrease is not worth the travel time delay and thus rejects the pickup decision. We thus introduce a parameter $Q_i.r$ for each rider Q_i, which presents Q_i's acceptable *money-to-time* rate. That is to say, Q_i supports the pickup of a new rider only when the ratio of the fare decrease to the travel time delay is larger than $Q_i.r$. The above constraint is expressed by Eq. (3.12).

$$\frac{\Delta f_i}{\Delta T_i} \geq Q_i.r \tag{3.12}$$

3.7 Pickup and Drop-off Interactions

In this section, we detail interactions among taxi riders, drivers, and the Cloud during pickup and drop-off events.

Pickup: Consider a rider whose request has been satisfied. She may wait the taxi on a street side or at a comfortable place, such as a coffee shop. As shown in the upper-left corner of Figure 17, the Cloud will send a reminder message to the rider's mobile phone when the assigned taxi is approaching her pickup point (e.g. within 30 meters based on the taxi's GPS readings). On receiving the reminder message, the rider will start paying attention to the taxi approaching and go standing by at the pickup point. Meanwhile, the rider's App will automatically turn on the Bluetooth in her phone, preparing a Bluetooth connection to the driver's phone. The purpose of such a connection lies in three aspects: 1) ensure a rider getting on the right taxi; 2) a rider can receive location updates from a driver's phone so as to save her

phone's battery from not using its own GPS; 3) receive the bill information from the driver when getting off. The connection is automatically established via a two-step hand-shake protocol. Each step involves one phone sending the reservation code to the other (the code was sent to both phones by the Cloud when the request was served). Since there may be multiple riders in one taxi, the driver's phone is set as a Bluetooth master maintaining a connection with each of the multiple slaves, as shown by the grey lines in Figure 17). The new rider confirms the boarding after actually getting on the taxi. The boarding confirmation will then be notified to the Cloud (via the driver's phone).

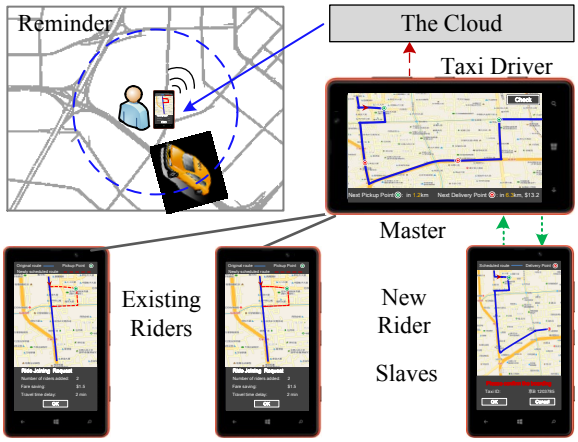


Figure 17 Interactions during a pickup event

Drop-off: Figure 18 shows interactions between the phones and the Cloud during a drop-off event. When the taxi reaches a destination point, the driver will press the "Check Out" button to trigger the following interaction. The driver's phone will send the bill information (shown by the left upper grey rectangle) to the corresponding rider's phone, as shown by the broken green arrow. The rider uses the phone to send a payment confirmation to the Cloud after paying the bill as shown by the blue arrow. The driver will confirm the payment (shown by the left lower grey rectangle) by clicking the "OK" button, which makes the driver's phone send a payment-received message to the Cloud as shown by the red broken arrow. The separate of two payment confirmations, i.e. one from the driver (i.e. the red arrow) and one from the rider (i.e.

the blue arrow), prevents the Cloud from being fooled by either party. For instance, without a payment-received message from the driver, the rider may send the payment confirmation to the server without actually paying the fare.

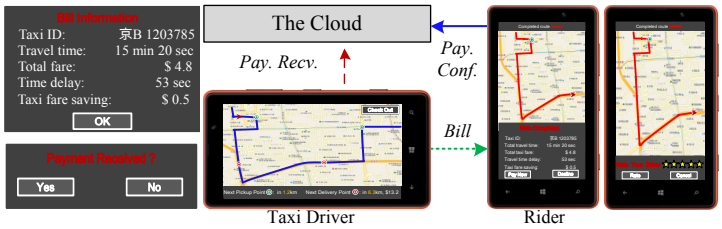


Figure 18 Interactions during a drop-off event

Once the transaction has been completed, the taxi driver and the taxi rider can rate each other. The right-most graph in Figure 18 shows the interface for a rider to rate the driver. The ratings of a driver or a rider is accumulated and maintained in the Cloud. When a driver or rider's rating is lower than a threshold, she is then no longer allowed to participate in our taxi-sharing system. We consider incorporating the credibility into the taxi searching and scheduling algorithms in the future, offering credible taxi drivers/riders a higher serving chance.

3.8 Experiments

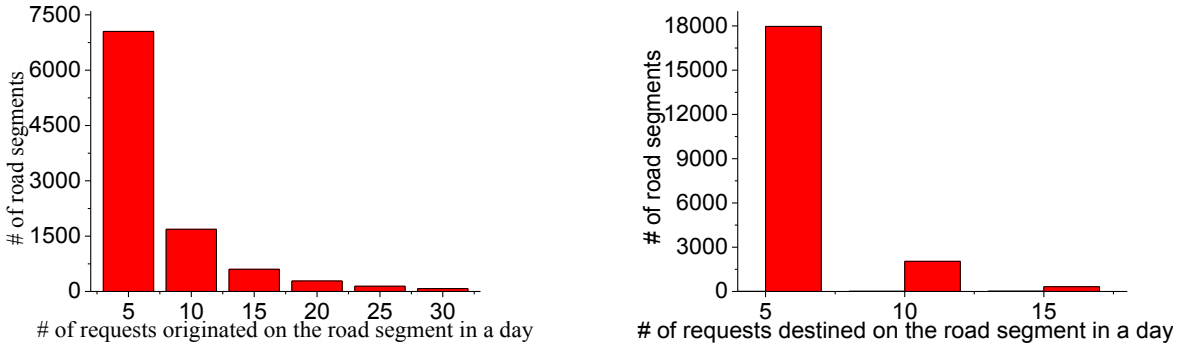
3.8.1 Setting

3.8.1.1 Data Set

Road networks: We perform the experiments using the real road network of Beijing, which contains 106,579 road nodes and 141,380 road segments.

Taxi Trajectories: The taxi trajectory dataset contains the GPS trajectory recorded by over 33,000 taxis during a period of 87 days spanning from March to May in the year of 2011. The total distance of the dataset is more than 400 million kilometres and the number of points reaches 790 million. After trip segmentation, there are in total 20 million trips, among which 46% are occupied trips and 54% are non-occupied trips. We map each occupied trip to the road network of Beijing using the map-matching algorithm proposed in [123]. Each trip then can be viewed as a query with windows

size equals to 0. Figure 19 shows the distribution of pickup and delivery points of the ride requests in the dataset over road segments in a day (long details, i.e. hot segments that are being the origin or destination of a large number of requets are not shown). It is clear that ride requests are distributed sparsely over the road network.



(c) Origins (d) Destinations

Figure 19 Distribution of ride requests over road segments

3.8.1.2 Experimental Platform

The historical trajectory dataset conceals rich information regarding 1) the distribution of the ride requests on the road network over time of day, and 2) the mobility patterns of the taxis. In order to validate our proposed system under practical settings, we mine the trajectory dataset to build an experimental platform, which generates a realistic ride request stream and initial taxi statuses for our experiments. We envision that this platform can be applied to many other relevant urban and transportation computation problems.

Ride request Stream: The goal is to generate real-time ride requests that are as realistic as possible. For this purpose, we first discretise one day into small time bins, denoted by b_i and denote all road segments by r_i. We assign all historical ride requests into time bins. Assume that the arrivals of ride requests on each road segment approximately follow a Poisson distribution during time frame f_j, where each frame has a fixed length spanning N time bins. Thus, we can learn λ_{ij}, i.e. the parameter of the Poisson distribution for road segment r_i during time frame f_j. Specifically, for

each road segment r_i, we count the number of ride requests that originated from r_i within time frame f_j, denoted by c_{ij}, and learn the distribution of the destination road segment of these ride requests, denoted by p_{ij}. Then we calculate λ_{ij} based on c_{ij} using Eq. (3.13) and generate a ride request stream that follows a Poisson process with parameterλ_{ij}.

$$\lambda_{ij} = c_{ij} / N \tag{3.13}$$

For each ride request Q generated in frame f_j with the origin road segment being r_i, the destination road segment is generated according to the distribution p_{ij}. $Q.pw.e$ and $Q.dw.e$ equals to $Q.t$, i.e. the birth time of the ride request. $Q.pw.l$ is calculated by applying a fixed window size. $Q.dw.l$ equals to the sum of $Q.pw.l$ and the average travel time between the origin and destination pair learned from the GPS trajectory dataset.

Note that the taxi GPS trajectory dataset only reveals the number of ride requests that got served. In reality there are also many ride requests unsatisfied and disappeared due to the shortage of taxis. To take such ride requests into consideration, we introduce a system parameter Δ, supposing the number of real ride requests is Δ times the number of request extracted from the trajectory dataset. Figure 20 shows the supposed number and the extracted number of ride requests fluctuating over time of a day, where the time frame is 1 hour and Δ=2.

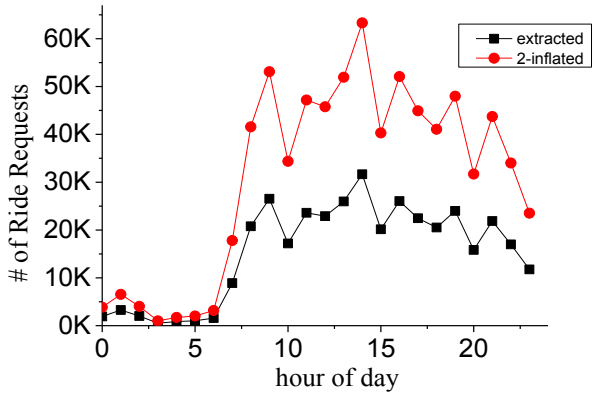


Figure 20 Inflated and extracted number of ride requests during a day

Initial Taxi Statuses: To keep the characteristics of the realistic scenario, we use the real taxi statuses by slicing the historical trajectories at a certain timestamp. Specifically, we select a date and choose a particular second of day as the timestamp when the experiment starts, denote it by t_s. We scan all the GPS records of the selected date to determine the initial states of taxis. A taxi status V is set to be occupied if it is recorded occupied crossing timestampt_s. The initial schedule of V can be initialized according to the record. A taxi V is set to be vacant if it is recorded vacant both just before and right after t_s. The concept of "just before" and "right after" is controlled by a temporal parameter, which is set to be 2 minute. All remaining taxis are then considered as not recorded and thus not used in the simulation.

The set of ride requests and initial states used in all validation experiments are generated with parameters listed in Table 3.

Table 3 Parameter Setting for Ride request Generation

Notation	Definition	Value
t_s	The start time of simulation	09:00
t_e	The end time of simulation	09:30
$\#taxi$	The number of taxis	2,980
$\#taxi_o$	The number of taxis occupied initially	2,072
ws	The window size	5 min
$l(b_i)$	The length of a time bin	5 min
N	The # of time bins in a frame	12

3.8.1.3 Framework

Based on the platform introduced above, we study two strategies in the searching algorithm (single-side and dual-side) and two strategies in the scheduling algorithm (first-fit and best-fit), resulting in four taxi-sharing methods. We compare the performance of these four methods with that of a non-taxi-sharing method as the number of requests (i.e., Δ) changes. We also test the performance of these four methods by changing the money-to-time rate parameter of the monetary constraints, and study the necessity of the schedule reordering step (i.e. considering different pickup

and drop-off orders) in the scheduling algorithm. As the travel time estimation technique had been extensively evaluated in [121, 122], we do not perform experiments on that again.

3.8.1.4 Baseline Methods

The *Non-Taxi-sharing method (NR)* forbids taxi-sharing and assumes that a vacant taxi moves to pick up the rider that it can pick up at the earliest time.

Taxi searching step: A taxi-sharing method is called *single-side* if the taxi searching algorithm retrieves taxis only from the origin side of a request; otherwise, it is *dual-side.*

Taxi scheduling step: A taxi-sharing method is called *best-fit* where the taxi scheduling process tries all candidate taxis returned by the taxi searching algorithm. Otherwise, is called *first-fit* if the scheduling process terminates immediately once it finds a taxi that satisfies the ride request.

Because the two choices can be made independently, we get the following four taxi-sharing methods: *Single-side and First Fit Taxi-sharing (SF)*, *Single-side and Best-fit Taxi-sharing (SB), Dual-side and First Fit Taxi-sharing (DF)*, *Dual-side and Best-fit Taxi-sharing (DB).*

The money-to-time rates of ride requests are assumed to follow an exponential distribution with a mean value.

3.8.1.5 Measurements

The performance of the taxi-sharing system is evaluated in two perspectives, namely effectiveness and efficiency. We first describe four effectiveness measurements as follows.

Relative Distance Rate (RDR): Define the distance of a ride request Q as the distance between its origin point $Q.o$ and its destination point $Q.d$. Denote by D_{SR} the sum of distances of ride requests that get satisfied and by D_V the total distance travelled by all taxis while being occupied in a taxi-sharing method. *RDR* is calculated by Eq. (3.14).

$$RDR = D_V / D_{SR} \tag{3.14}$$

RDR evaluates the effectiveness of taxi-sharing by measuring how much distance is saved compared to the case where no taxi-sharing is practiced.

Satisfaction Rate (SR): is the ratio of the number of ride requests that get satisfied to the total number of ride request (exclude ride requests that are already served by taxis at the initial state in the ride request counting). *SR* is another crucial criterion measuring the effectiveness of the taxi-sharing system.

Taxi-sharing Rate (TR): is the percentage of ride requests participating in taxi-sharing among all satisfied ride requests.

Fare Saving Rate (FSR): is the average saving percentage in taxi fare of riders whose participate in taxi-sharing.

Now we introduce following efficiency measurements.

Number of Road Nodes Accessed Per Ride request (#GNAPR): is the number of accessed road network nodes per ride request.

Number of Road Nodes Accessed Per Ride request (#GCAPR): is the number of accessed grid cells per ride request.

Number of Taxis Accessed Per Ride request (#TAPR): This measurement records how many taxis per ride request are accessed by the scheduling module.

RNAPR, GCAPR, TAPR are machine-independent indicators for computation cost of the system since the majority of on-line computation is done in the scheduling process.

Execution Time Per Ride request: is the CPU time spent for serving each ride request. It consists of taxi searching time, (i.e. time elapsed between step ② and ④ in Figure 7) and taxi scheduling time (time elapsed between ⑤ and ⑦ in Figure 7).

3.8.2 Results

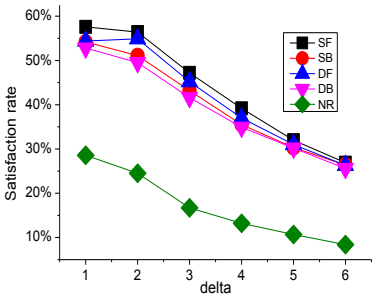
Table 4 lists default values for parameters used in experiments.

Table 4 Default values of parameters used in experiments

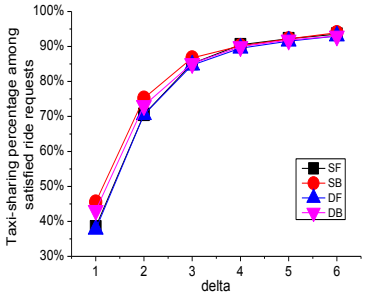
Definition	Value
Beijing Map Size	32 *40 km^2
The size of grid (i.e. number of grid cells)	30*30
Schedule reordering before insertion	no
Taxi fare per kilometre	¥2
Mean of the money-to-time rate distribution	¥0/min

Effectiveness: Figure 21 (a) shows SR, i.e. the satisfaction rate, of all methods as Δ changes. All methods show a decline in SR as the number of ride requests increases. All flavours of taxi-sharing methods have a considerably higher satisfaction rate (about 23% higher on average) than the NR method for all delta values. The difference in the satisfaction rate among taxi-sharing methods is insignificant as no particular technique is proposed for minimizing the satisfaction rate.

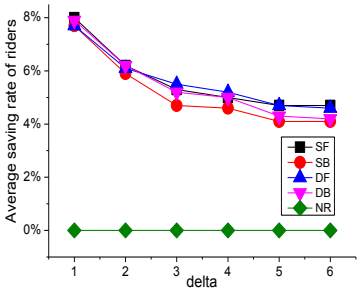
Figure 21 (b) shows TR, the percentage of ride requests participating in taxi-sharing among all satisfied ride requests, for all taxi-sharing methods. Not surprisingly, TR surges as Δ increases. This is because taxi-sharing opportunities are likely to rise as the number of taxi ride request increases. Consequently, more ride requests can be satisfied via taxi-sharing. But since the total number of ride requests increases even faster, the satisfaction rate still drops, as illustrated by Figure 21 (a).



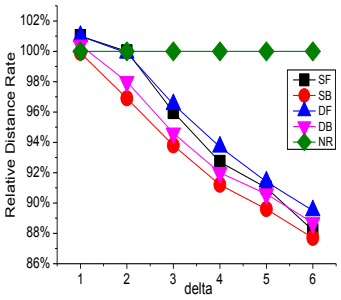
(e) Satisfaction rate vs. delta



(f) Taxi-sharing rate vs. delta



(g) Fare saving rate vs. delta



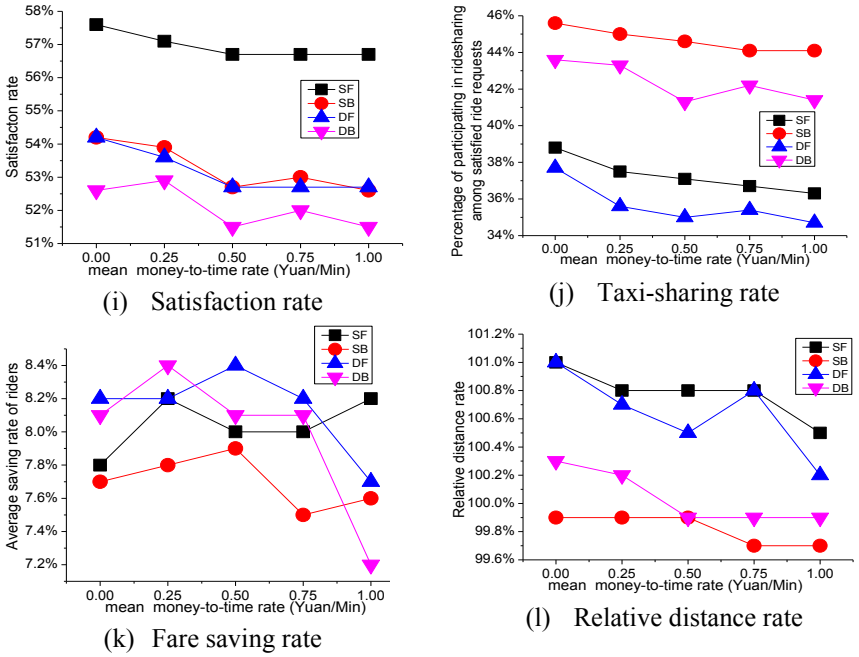
(h) Relative distance rate vs. delta

Figure 21 Performance in effectiveness measurements of different methods

To calculate the fare of riders in the experiments, we instantiate the fare calculation function as follows: the fare charged by a taxi driver is linear to the distance travelled by the taxi, i.e. the product of the distance travelled and fare per unit distance. Figure 21 (c) shows that *FSR*, the average fare saving of riders who participate in taxi-sharing, drops as Δ increases. The average cost of this amount saving is about 5.08 minute delay in travel time. We believe that most riders are willing to tolerate this amount of delay, especially under the high request demand scenarios in which this taxi-sharing system is most likely to be useful.

Figure 21 (d) shows *RDR* steadily drops as parameter Δ increases. Again, this is likely because as the number of ride requests increases, more ride requests can share partial trips with each other and thus more distance the taxi-sharing methods save. The *SB* taxi-sharing method outperforms other methods, since *SB* reduces the increase in travel distance most. The *DB* taxi-sharing method slightly trails *SB* method as the taxi searching step of it explores fewer grid cells. Two first-fit based taxi-sharing methods show clearly higher relative distance rate. From the picture, we can see that taxi-sharing methods save up to 12% in travel distance, depending on delta. Given the fact that there are 67,000 taxis in Beijing and each taxi runs 480 km per day (learned from the dataset), the saving achieved by taxi-sharing here means over 1.5 billion kilometres in distance per year, which equals to 120 million litre of gas per year (Supposing a taxi consumes 8 liter of gasoline per 100km) and 2.2 million of carbon dioxide emission per year (supposing each litre of gas consumption generates 2.3 kg of carbon dioxide).

Figure 22 (a), (b), (c), (d) shows *SR, TR, FSR* and *RDR* of taxi-sharing methods for different mean values of the money-to-time rate of ride requests, respectively, when $\Delta = 1$. All measurements except *FSR* show a clear decrease tendency as the mean money-to-time rate increases. When the mean money-to-time rate increases from ¥0.25/min to ¥0.5/min, the decrease is most significant.



(i) Satisfaction rate (j) Taxi-sharing rate

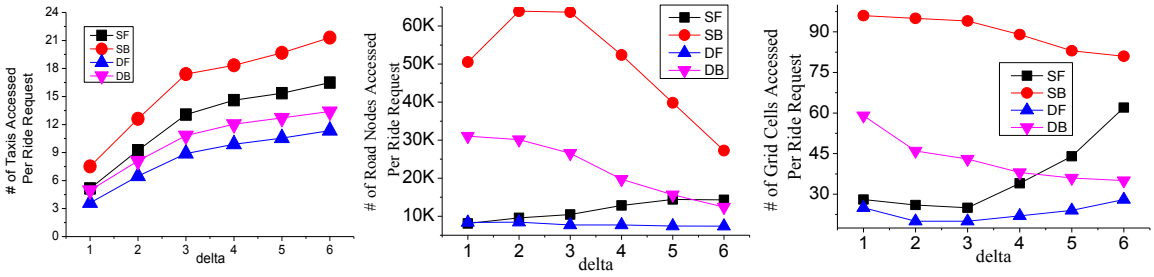
(k) Fare saving rate (l) Relative distance rate

Figure 22 Performance in effective measurements vs. money-to-time rate

Efficiency: Tested on a single server with 2.67GHz CPU and 16GB RAM (using a single thread), the average taxi searching time and scheduling time is 0.15 ms and 10.33 ms, respectively. If we take the CPU time on travel time estimation (about 350 ms per OD pair using the same machine [121]) and communication time between smart phones and the Cloud into consideration, the total response time of a ride request is about 40~50 seconds. This time can be reduced largely if we implement the travel time estimation method using parallel techniques.

We also test the efficiency of the system using machine-independent measurements. The three sub-graphs of Figure 23 show the number of taxis accessed per ride request, the number of road nodes accessed per ride request, and the number of grid cells accessed per ride request, respectively,

for different taxi-sharing methods under different Δ. It is clear from the pictures that all taxi-sharing methods do not show sharp increase in computation cost as Δ increases. It is also obvious that the computation cost of the *DB* taxi-sharing method is significantly smaller than that of *SB* taxi-sharing method. Especially when Δ≥ 4, the computation cost of the *DB* method is even smaller than that of the *SF* method. The result of Figure 21 and Figure 23 together validate our motivation for the dual-side taxi searching algorithm. That is, the dual-side searching indeed incurs small increase in travel distance in exchange for the significant decrease in computation cost.



(m) #TAPR vs. delta (n) #RNAPR vs. delta (o) #GCAPR vs. delta

Figure 23 Computation cost in terms of node access per ride request

Necessity of the Schedule Reordering: Figure 24 shows the average execution time (excluding the time spent for the travel time estimation) per ride request under different values of Δ when using the *DB* taxi-sharing method with and without the schedule reordering before insertion. The execution time per ride request is about 20% longer on average when the schedule reordering is performed.

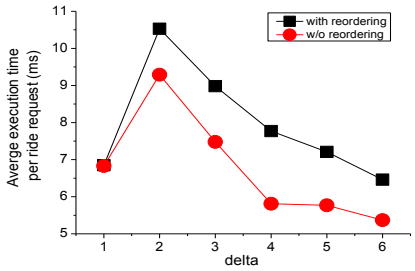


Figure 24 Time cost of schedule reordering

Meanwhile there is almost no change in all effectiveness measurements including satisfaction rate, relative distance rate, etc. From the results, we also learned that in practice it is extremely rare that the optimal insertion requires the schedule reordering. Although the execution time per ride request remains a reasonable small value with the schedule reordering step, there is still no incentive to do so in practice.

3.9 Discussion

We have proposed and developed a mobile-cloud based real-time taxi-sharing system. We presented detail interactions between end users (i.e. taxi riders and drivers) and the Cloud. We validated our system based on a GPS trajectory dataset generated by 33,000 taxis over 3 months, in which over 10 million ride requests were extracted. The experimental results demonstrated the effectiveness and efficiency of our system in serving real-time ride requests. Firstly, our system can enhance the delivery capability of taxis in a city so as to satisfy the commute of more people. For instance, when the ratio between the number of taxi ride requests and the number of taxis is 5, our proposed system served additional 22% ride requests compared with no taxi-sharing. Secondly, the system saves the total travel distance of taxis when delivering passengers, e.g. it saved 12% travel distance with the same ratio mentioned above. Supposing a taxi consumes 8 liters of gasoline per 100 km and given the fact learned from the real trajectory dataset that the average travel distance of a taxi in a day in Beijing is about 480 km, the system can save over one third million liter of gasoline per day, which is over 120 million liter of gasoline per year (worth about 150 million dollar). Thirdly, the system can also save the taxi fare for each individual rider while the profit of taxi drivers does not decrease compared with the case where no taxi-sharing is conducted. Using the proposed monetary constraints, the system guarantees that any rider that participates in taxi-sharing saves 6% on average. In addition, the experimental results justified the importance of the dual-side searching algorithm. Compared to the single-side taxi searching algorithm, the dual-side taxi searching algorithm reduced the computation cost by over 50%, while the travel distance was

only about 1% higher on average. The experimental results also suggests that reordering the points of a schedule before the insertion of the new ride request is not necessary in practice for the purpose of travel distance minimization.

In the future, we consider incorporating the creditability of taxi drivers and riders into the taxi searching and scheduling algorithms. Additionally, we will further reduce the travel distance of taxis via ridesharing.

Chapter 4

Analysis and Evaluation of the Slugging Form of Ridesharing

4.1Introduction

Transportation problems, such as traffic jams, finding parking slots, hailing a taxi during rush hours, are long-existing headaches in cities, especially those with a large population. These problems negatively affect the environment, the economy, and more importantly average peoples' daily lives.

Different methods have been mainly proposed to tackle these problems separately. For example, extending the road network is one common approach to tackle traffic jams; sensors which can detect the availability of parking spaces [4] are installed to help drivers find open parking slots more quickly. However, those solutions often require additional construction or new equipment added to the existing infrastructures and thus are often expensive to implement. In addition, their benefits are usually limited to the specific corresponding problem.

One reason for the above transportation problems is that the passenger seats of vehicles are under-utilized. Thus, we study ridesharing as a promising means to improve the utilization of vehicle ridership and thus reduce the number of cars on the road.

Ridesharing practices have a variety of characteristics. For example, ridesharing can be either dynamic or static. Dynamic ridesharing arranges trips on a very short notice. By contrast, static ridesharing arranges trips that are known in advance, usually hours or a day or two before the departure time. Ridesharing can arrange either recurring or ad-hoc trips. Also, ridesharing can either change or keep the route of the original trips of drivers. (In case routes are kept, riders need to get on and off the driver's car at the origin and destination locations of the driver instead of their

own.) Riders may share the cost with the driver or not. Table 5 summarizes the characteristics of some of the most common ridesharing applications.

Table 5 Characteristics of some of the most common ridesharing applications

Ridesharing Applications	Characteristics			
	Dynamic	Recurring Trip	Route Change	Cost Sharing
taxi ridesharing	yes	no	yes	yes
hitchhiking	yes	no	no	no
carpooling	no	yes/no	yes	yes
slugging [18]	yes/no	yes/no	no	no/very-low

Here we are interested in one particular ridesharing form, i.e. slugging. In slugging a passenger walks to the driver's origin, boards at the driver's departure time, alights at the driver's destination, then walks from there to the passenger's own destination. Thus slugging involves two modes of transportation, car and walking. Since slugging does not change any spatio-temporal aspect of the drivers' original trips, slugging is the simplest form of ridesharing in the sense of bringing minimum disruptions to the drivers. Thus it can be offered at minimum or no-cost to the riders. Compared to other forms of ridesharing where route change is allowed, e.g. taxi ridesharing [77], slugging avoids unnecessary complications such as complex fare mechanism or ridesharing-incurred travel time delay for drivers (e.g. due to unexpected congestion encountered on the way to some pickup). Thanks to its simplicity, slugging has already become a common transport mode in some of the busiest traffic areas in the North America, e.g. auxiliary interstate highways around urban areas such as Washington D.C., Bay area, Houston, and other cities [6, 13].

Though currently slugging is mainly used for regular commute trips, we envision that it can also be applied to ridesharing scenarios that involve mostly one-time casual trips. For example, consider a ridesharing website where travelers post their trips scheduled in the near future. When posting their trip, travelers may announce their roles in ridesharing: drivers, passengers, or both (i.e. travelers who have a car can leave the role to be determined by the website). The website will compute a slugging plan to group these travelers and decide the driver and passengers for each

group. The only attached string for a passenger is that she needs to walk to the origin location of the driver's trip before the driver departs, and she needs to walk from her driver's destination to her own destination. Drivers are willing to accept such a ride for a various reasons, such as environmental-friendliness, companionship, the privilege of driving on HOV lanes, reduced or waived toll on highways, small payment, etc.

The increasing popularity of bike sharing programs indicates that people are open to alternative modes of transportation, particularly the ones like slugging that involve physical activity (i.e. walking). The motor industry is also actively promoting shared services like slugging, as stated in the "Blueprint for Mobility" vision recently released by Ford company.

To the best of our knowledge, our work is the first one to study slugging from a computational perspective. We define and study the basic slugging problem and its variants that are constrained by the vehicle capacity and travel time delay. We also discuss the dynamic version of the slugging problem. The experimental results show that our proposed heuristics achieve 59% saving in vehicle travel distance. Given the size of our real data set is 39 thousand trips and the average distance of a trip in the data set is 6.3 kilometres, the saving equals to 144,963 kilometres, which means the reduction of over 4.5 thousand gallons of gasoline and 71 tons of carbon dioxide emission.

In summary, the contributions of this section include:

- We formalize the slugging problem using a graph abstraction. We propose a quadratic algorithm to solve the slugging problem.
- We define a generalization of the slugging problem and prove its NP-completeness.
- For the variants of the slugging problem that are constrained by the vehicle capacity and travel time delay, we prove their NP-completeness and propose effective heuristics. Via extensive experiments, we demonstrate that the proposed heuristics have near-optimal performance in terms of the saving in vehicle travel distance.

- We also consider the dynamic slugging problem and evaluate it via experiments; in the dynamic problem the trips are announced incrementally.

The remainder of the section is organized as follows. In Section 4.2, we review existing literature related to our work. Section 4.3 formally defines and studies the slugging problem, its generalization, its constrained variants, its dynamic version, and heuristics for the intractable variants. We evaluate the proposed heuristics in Section 4.4.

4.2Related Works

In this section we review existing works on three problems that are relevant to slugging, i.e. taxi-ridesharing, carpooling and the dial-a-ride. Similar to slugging, all these problems are transportation problems that involve pickups and drop-offs. Unlike slugging where passengers change their origin and destinations in order to join the trip of drivers, in all three problems, drivers change their route in order to pick up and deliver the passengers. Both taxi ridesharing and carpooling are specific forms of ridesharing. The difference is that each driver in carpooling usually is associated with her own trip, while in taxi ridesharing this is not the case. Also taxi ridesharing usually needs appropriate pricing mechanisms to incite taxi drivers. The dial-a-ride problem slightly differs from carpooling as all vehicles start a trip and return to the same location called the depot.

4.2.1Taxi Ridesharing

There have been a number of works on the taxi ridesharing application [77, 78, 86, 107]. These works modelled the taxi ridesharing problem by considering different constraints. In contrast to slugging, the routes of driver trips, i.e. taxis in this case, change to accommodate passengers. Among these works, some (see [107]) only considered vehicle capacity constraints, while the rest also considered time window constraints, i.e. travelers need to depart and arrive in given time intervals. [78] is the only paper that models monetary constraints, which are used to guarantee monetary incentives for both taxi drivers and taxi riders. These works on taxi ridesharing mainly

concern the efficiency and scalability of ridesharing, i.e. how fast a query can be answered and how many queries the system can handle. In contrast, we focus on the effectiveness of slugging as a whole, e.g. the saving in vehicle travel distance, while the existing works on taxi ridesharing often consider the effectiveness of ridesharing from the perspective of a single request, e.g. reducing the increase in vehicle travel distance for every new request [77].

4.2.2Carpooling

There have been many works on modelling and analysing the traditional carpooling problem where drivers need to change their routes due to ridesharing. In [19], the authors modelled a carpooling problem and proposed an exact method based on Lagrangean column generation to solve it optimally. Since the carpooling problem is NP-hard, the exact approach practically only works for small instances of the carpooling problem, where there are at most a few hundred trips. For large instances with hundreds of thousands trips, many heuristics have been proposed [14, 109]. These heuristics are applied to compute the best route of a vehicle for a given set of requests, since the route of drivers is allowed to change. As such route changes do not occur in slugging, these heuristics are not applicable.

Despite being a sibling of the carpooling problem, the slugging problem has so far drawn little attention from researchers. There have been some reports on the current state of slugging operations (see [28]). But our work is the first formal study of slugging from a computational viewpoint.

4.2.3Dial-A-Ride Problem (DARP)

The *Dial-A-Ride Problem (DARP)* [16], a.k.a. the Vehicle Routing Problem with Time Windows in the operation research literature, is closely relevant to the carpooling problem. The *DARP* can be considered the carpooling problem with additional restrictions (e.g. all vehicles are required to start any trip from a depot location and return to the depot after the trip). In contrast to slugging, vehicle routes are manipulated to accommodate passengers' origin and destination locations. *DARP* is proved to be NP-hard. Cordeau et al. summarizes the state-of-the-art heuristics for *DARP* [33].

4.3 Slugging

We introduce the concept of slugging in Sec. 4.3.1. We formally define the basic slugging problem in Sec. 4.3.2. Next we introduce and discuss the vehicle-capacity constrained slugging problem in Sec. 4.3.3, and the delay bounded slugging problem in Sec. 4.3.4. Then we describe the slugging problem with both constraints and propose heuristics for it in Sec. 4.3.5. Finally, we discuss the dynamic slugging problem and its parameters in Sec. 4.3.6.

4.3.1 Preliminaries

In slugging, some travelers abandon their original trips and join the trip of other travellers, the drivers, without asking the drivers to change their route or their departure time. To be more specific, consider two travellers A and B, and their respective trips T_A and T_B, each of which is described by an origin destination pair and a start time at which the traveller intends to depart. Assume that traveller A abandons her trip and joins B's trip. In this case we say that T_A is *merged into* T_B. More specifically, traveller A executes her new trip as follows: at the start time of T_A she walks to the origin location of trip T_B, then she waits until the start time of T_B (if A arrives later than the start time of T_B then she cannot join T_B), she shares the ride with B, she alights at the destination of T_B and finally she walks from there to her own destination. Clearly, the only impact that traveller A has on trip T_B is the occupation of one seat in B's vehicle. In other words, there is no disruption to any spatio-temporal aspect of T_B.

In the above example, there is only one traveler associated with each trip. In general, each trip can be associated with a party of multiple travelers who cannot be separated during the trip (assuming that the size of the party is always smaller than the number of seats in a vehicle).

As shown in the above example, one necessary condition for trip T_i to be able to be merged into trip T_j is that the travellers of trip T_i can walk from the origin of T_i at the start time of T_i and arrive at the origin of trip T_j before the start time of T_j (assuming a constant walking speed and taking the

shortest path). Consider a set of trips $S_T = \{T_1, T_2, \ldots, T_m\}$ where the travelers of each trip T_i announce their willingness to serve as: driver, or passenger, or both. Then for each trip pair T_i and T_j, where the travelers of T_i have announced their willingness to be passengers, and the travelers of T_j have announced their willingness to be drivers, we can compute whether or not T_i can be merged into T_j. To do that, a preprocessing stage is performed. At this stage, a map is used to compute the shortest path between the respective origins. Specifically, for such a trip pair (T_i, T_j), the shortest path between the origins of the two trips is computed. Based on the calculated shortest path, a presumed walking speed, and the start times of T_i and T_j, we can readily determine whether or not trip T_i can be merged into T_j. If so, we say that pair (T_i, T_j) is a *mergable pair* where T_i is a *passenger trip* and T_j is a *driver trip*. For a mergeable pair (T_i, T_j), the shortest path between the destinations of T_i and T_j is also calculated in order to determine the travel time delay for the passenger trip T_i. The travel time delay for passenger trips imposes a natural constraint on the slugging problem, which will be discussed further in Sec. 4.3.4 and 4.3.5.

Now that we have defined a mergeable pair, for a given set of trips, consider the set of all mergeable pairs represented as a graph *S*. Assuming that the trip start-times are distinct, we observe that *S* possesses the following two properties.

First, *S* is *acyclic*. Suppose there exists a cycle of mergeable pairs (T_{i_1}, T_{i_2}), (T_{i_2}, T_{i_3}), …, (T_{i_n}, T_{i_1}) in *S*. Mergeable pair (T_{i_n}, T_{i_1}) means that the start time of T_{i_n} is smaller than that of T_{i_1}. However, the first *n*-1 pairs of the cycle collectively tell us that the start time of T_{i_1} should be smaller than that of T_{i_n}. Contradiction. In other words, *S* is acyclic because the start-times of the trips on a path in *S* are increasing.

Second, *S* is *transitive* (i.e. if $(T_i,\ T_j) \in S$ and $(T_j,\ T_k) \in S$), then $(T_i,\ T_k) \in S$. If the travelers of T_i can arrive at the origin of T_j before the start time of T_j, and the travelers of T_j can arrive at the origin of T_k before the start time of T_k, then the travelers of T_i definitely can arrive at the origin of

T_k before the start time of T_k as well by: first arriving at the origin of T_j and then taking the same path used by the travelers of T_j to the origin of T_k; this assumes that all travelers have the same walking speed.

4.3.2 Basic Slugging Problem

Slugging is a graph problem. We formulate it as follows.

Definition 1 *A slugging graph* $G = (V, E)$*, is a directed acyclic graph where* $V = \{T_1, T_2, \dots, T_m\}$ *is a set of trips and E is set of directed edges between nodes that is transitive, i.e. if* $(T_i, T_j) \in E$ *and* $(T_j, T_k) \in E$*, then* $(T_i, T_k) \in E$.

Note that a node in a slugging graph may not have any incident edges. A node with no incident edge can exist as it represents a trip that cannot be merged into any other trip, or into which no other trip can be merged. For example, a trip geographically bounded in the North Eastern corner of a city may become such a disconnected node if all other trips are bounded in the South Western corner of the city, and they all start at approximately the same time.

A slugging graph indicates which trips can be merged into others. However, although a trip can be merged into multiple other trips, in a concrete slugging plan it is merged into only one other trip. In other words, a slugging graph gives the possible pairs of trips that can be combined, whereas a slugging plan gives an actual combination that will be executed in practice. So, based on a slugging graph, a slugging plan can be constructed. Intuitively, a slugging plan is a subgraph of the slugging graph that gives the driver and the passengers of each car.

Definition 2 *Given a slugging graph* $G = (V, E)$*, a slugging plan* $G_S = (V, E_S), E_S \subseteq E$*, is a subgraph of G that satisfies the following conditions: (i)* $\forall (T_i, T_j) \in E_S$*, there is no* $k \neq j$ *such that* $(T_i, T_k) \in E_S$*; and (ii)* $\forall (T_i, T_j) \in E_S$*, there does not exist k such that* $(T_k, T_i) \in E_S$.

Intuitively, condition (i) states that any trip T_i can be merged into at most one other trip. Condition (ii) states that a trip T_i can be merged into another trip T_j only if there is no other trip T_k that has been merged into T_i. These constraints precisely reflect the nature of the slugging problem: each trip is either a ridesharing provider, i.e. providing a car to be shared with other riders, or a ridesharing consumer, i.e. taking exactly one ride provided by a provider.

Figure 25 gives an illustrative example of slugging plans. Subfigure (a) shows a slugging graph of four trips. Subfigures (b) (c) (d) (e) show all slugging plans that are maximal, i.e. cannot include more edges. For instance, consider the slugging plan shown in subfigure (b). Given that (T_4, T_3) already exists, neither edge (T_4, T_1) nor edge (T_4, T_2) can be added because the addition violates Condition (i), and neither edge (T_3, T_2) nor edge (T_3, T_1) can be added because either addition violates Condition (ii).

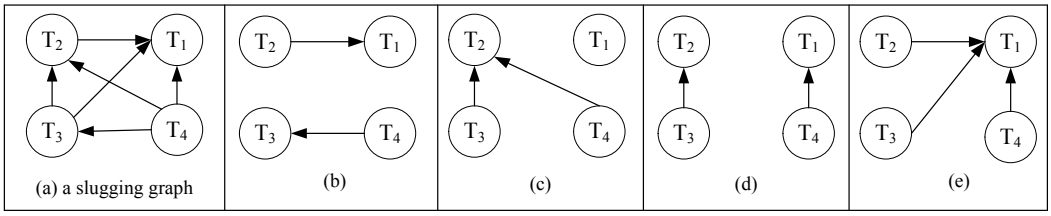


Figure 25 An illustrative example of slugging plans

A mergable pair (T_i, T_j) in a slugging plan means that T_i is merged into T_j. That is to say, T_i is simply eliminated while there is no change to T_j other than the fact that the number of passengers in T_j 's vehicle is increased. Therefore the benefit of merging T_i into T_j only depends on the passenger trip T_i and thus can be measured by some attribute of T_i, e.g. the vehicle travel distance that is saved. In other words, the benefit of merging T_i into another trip is independent of the other trip. The implication is that if an edge is labeled by the benefit of merging the two trips at its endpoints, then all the edges exiting a node have the same benefit. Formally, we define the benefit of a slugging graph as follows.

Definition 3 *A slugging graph $G = (V, E)$ is called benefit-labeled if each edge $(T_i, T_j) \in E$ is associated with a label $B(T_i, T_j) \in \mathbb{R}^+$, referred to as the benefit of edge (T_i, T_j), and the benefits of all edges outgoing of the same node are identical, i.e. $\forall T_i, T_j, T_k$ such that $(T_i, T_j) \in E$ and $(T_i, T_k) \in E$, $B(T_i, T_j) = B(T_i, T_k)$.*

A straightforward example of a benefit function is the constant function $B(T_i, T_j) = 1$ for any mergeable pair (T_i, T_j). Intuitively, this benefit function measures the number of trips saved by ridesharing. Another example of a benefit function is: $B(T_i, T_j)$ equals to the vehicle travel distance of trip T_i. Intuitively, this benefit function measures the saving in vehicle travel distance.

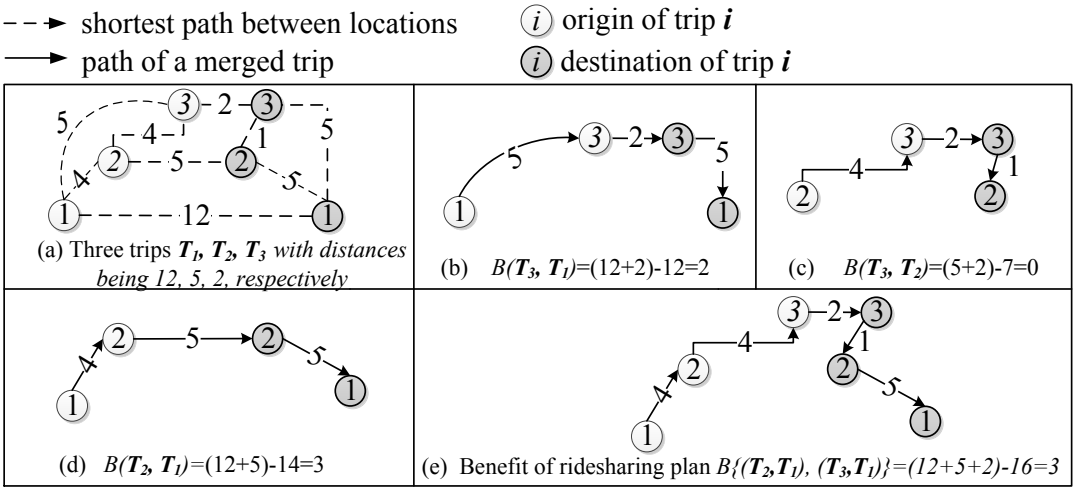


Figure 26 An example of a benefit function for a ridesharing form in which driver trips are changed

Definition 3 essentially says that the benefit of a mergeable pair is independent of the driver trip. Note this characteristic is unique to slugging and is not applicable to other ridesharing forms. For example, if we consider a ridesharing form where the route of driver trips can be changed, such as taxi ridesharing, then a benefit function B that measures the saving in the total travel distance is dependent on the driver trip. Figure 26 shows an illustrative example of this case. Figure 26 (a) shows three trips with their travel distances, and the distances between the origins and destinations of

these trips. Figure 26 (b) and (c) show a trip after merging T_3 into trip T_1 and T_2, respectively, resulting $B(T_3, T_1) = 2$ and $B(T_3, T_2) = 0$. In other words, since the passenger is picked up at her origin and dropped off at her destination, the total saving in travel distance depends on the driver's origin and destination.

The next definition gives the benefit of a ride-sharing plan as the total benefit of its edges.

Definition 4 *Given a slugging graph $G = (V, E)$ that is benefit-labeled, the benefit of a slugging plan $G_S = (V, E_S)$, denoted by $B(G_S)$, is the sum of the benefits of the edges in E_S. That is to say, $B(G_S) = \sum_{(T_i, T_j) \in E_S} B(T_i, T_j)$.*

Definition 4 is also applicable to slugging only, but not to other ridesharing forms. To illustrate this point, consider again the example shown by Figure 26. The benefit of merging T_2 into T_1 is 3, as shown by Figure 26 (d); and the benefit of merging T_3 into T_1 is 2, as shown by Figure 26 (b). However, as shown by Figure 26 (e), the benefit of slugging plan $\{(T_2, T_1), (T_3, T_1)\}$ is 3 rather than 5, which is the sum of the benefit of the two pairs in the plan.

Problem 4.1 *Given a slugging graph $G = (V, E)$ that is benefit-labeled, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a slugging plan and has the maximum benefit. We refer to this as the Slugging Problem (SP).*

Theorem 4.1 *SP can be solved in $O(|V|^2)$ time.*

Proof A trip $T_i \in V$ is called a *sink trip* if its node has no outgoing edges. Due to the fact that G is acyclic and transitive, for each non-sink trip T_i, there exists at least one sink trip T_s such that $(T_i, T_s) \in E$.

Now we can construct the optimal slugging plan for *SP* using the algorithm as shown by Figure 27. The G_S in Figure 27 merges each trip T_i that is not a sink trip into any sink trip T_k such that $(T_i, T_k) \in E$.

Algorithm 1: Quadratic Algorithm for SP

Data: the slugging graph $G = (V, E)$ that is benefit-labeled
Result: the ridesharing plan G_S with the maximum benefit

```
1  passengerTrips ⟵ ∅
2  driverTrips ⟵ ∅
3  for edge (T_i, T_j) ∈ E do
4  |  passengerTrips ⟵ passengerTrips ∪ {T_i}
5  |  driverTrips ⟵ driverTrips ∪ {T_j}
6  sinks ⟵ driverTrips − passengerTrips
7  G_S ⟵ ∅
8  for trip T_i ∈ V do            /* iterate all trips in the set */
9  |  if T_i ∉ sinks then
10 |  |  Pick any sink trip T_k ∈ sinks such that (T_i, T_k) ∈ E and add edge
   |  |  (T_i, T_k) to G_S
11 return G_S
```

Figure 27 Quadratic algorithm for SP

It is not hard to see that the constructed $G_S = (V, E_S)$ is indeed optimal. First G_S is constructed such that each passenger trip has been merged into some driver trip. And since that the benefit of merging a passenger trip is the same regardless which driver trip the passenger trip is merged into, therefore, the benefit of G_S is maximum

Let us consider the time complexity of Algorithm 1. As shown by Line 1~6, trips that are sinks can be identified in $O(|E|)$ time. From Line 8~10, the slugging plan is calculated. Since there are at most $O(|V|)$ non-sink trips, and for each non-sink trip it takes at most $O(|V|)$ time to find a sink trip into which the non-sink trip can be merged, then the time complexity of Line 8~11 is $O(|V|^2)$ as well. Since $|E|$ is $O(|V|^2)$, the time complexity of Algorithm 1 is $O(|V|^2)$. □

The transitivity of the slugging graph relies on the assumption that travelers walk at the same speed. If we relax this assumption, then the slugging graph is no longer transitive. This relaxation leads to a generalization of *SP* in which the graph is only acyclic. We prove next that this generalization of *SP* is NP-complete.

Problem 4.2: *Given a directed acyclic graph $G = (V, E)$ where V is a set of trips and E is the set of edges that is benefit-labeled, and a number $R \in \mathbb{R}^+$, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that*

is a slugging plan with the benefit at least R. We refer to this problem as the Generalized Slugging Problem (GSP).

Theorem 4.2: *GSP is NP-complete.*

Proof: First, it is easy to see *GSP* is in NP. Now, we prove *GSP* is NP-hard by reducing the set cover problem to *GSP*.

The set cover problem is well-known NP-hard. It is defined as follows: given a set U of n elements, a family of subsets of U, $\{S_1, S_2, \dots, S_m\}$ and a integer k, the question is whether there exists a set of at most k of these subsets whose union equals to U. If the answer is yes, the problem has a set covering of size k.

Given an instance of set covering problem, i.e. a universe $U = \{1,2, \dots n\}$ and a family of m subsets $S_1, S_2, \dots, S_m$ of U, we can build an instance of *GSP* as follows. First we construct the graph $G = (V, E)$ as follows: We define V = $\{T_1, T_2, \dots, T_n, T_{S_1}, T_{S_2}, \dots, T_{S_m}, T_{sink}\}$ and construct E as follows. If $i \in S_j$, add an edge (T_i, T_{S_j}) to E for all $i = 1,2, \dots, n$ and $j = 1,2, \dots, m$, and add an edge $\left(T_{S_j}, T_{sink}\right)$ to E for all $j = 1,2, \dots m$. Note that E is indeed acyclic. We define benefit function B as a constant function $B\left(T_i, T_j\right) = 1$ for all $\left(T_i, T_j\right) \in E$. We also define benefit threshold $R = n + m - k$. Now we show that the set cover problem has a set covering of size k iff there is a subgraph of G that is a slugging plan and has a benefit as least of R.

First, assume that the set cover instance admits a set covering of size k, denoted by $\mathbb{C}$, we will now construct a subgraph of G, denoted by G_S, that is a slugging plan and has a benefit as least of R, i.e. $n + m - k$, as follows: start with an empty subgraph G_S; for each node $T_i, i \in U$, choose another node T_{S_j} such that $i \in S_j$ and $S_j \in \mathbb{C}$, then add edge (T_i, T_{S_j}) to the edge set of G_S. Since $|U| = n$, the benefit of RP increases by 1 for n times. For each set $S_j \notin \mathbb{C}$, add edge (T_{S_j}, T_{sink}) to the edge set of G_S, so $B(G_S)$ increases by 1 for at least $m - k$ times. G_S is a legitimate slugging plan since each $T_i, i \in U$ and each $T_{S_j}, S_j \notin \mathbb{C}$ is chosen to be passenger trips only while each

$T_{S_j}, S_j \in \mathbb{C}$ and T_{sink} is chosen to be driver trips only, and no trip is merged into more than one other trips. The benefit of G_S is at least $n + m - k$.

Conversely, assume that there is a slugging plan $G_S = (V, E_S), E_S \subseteq E$ of benefit at least $n + m - k$, we now prove that there is a set covering of size k by proof of contradiction. Suppose there is no set covering of size k. Since for each node $T_i, i \in U$, it can contribute at most 1 to the benefit of G_S, and all n T_i's collectively contribute at most n to the benefit of G_S. Let us first assume that all T_i's are contributing. Denote by $\mathbb{C}$ the set of S_j's such that edge $(T_{S_j}, T_{sink}) \notin G_S$. Clearly $\mathbb{C}$ is a set cover. Since any cover size is larger than k thus $|\mathbb{C}| > k$, then there are less than $m - k$ nodes T_{S_j} are free to merged into node T_{sink}, i.e. contribute 1 to the benefit of G_S. Thus $B(G_S) < n + m - k$. Contradiction. Now assume that not all n T_i's contribute to G_S, say edge (T_i, T_{S_j}) is removed from G_S, the removal may or may not set T_{S_j} free, i.e. allow (T_{S_j}, T_{sink}) add to G_S. Even (T_{S_j}, T_{sink}) is added to G_S, since (T_{S_j}, T_{sink}) is previously removed from G_S, and thus the benefit of G_S will not increase. Contradiction remains. Therefore, there must be a set cover of size of k.□

4.3.3 Capacitated Slugging

The basic slugging problem may work well for the case where vehicles have a large number of seats, such as (mini)buses. The reason is that the problem does not constrain the number of passengers that a driver can take. The problem becomes more general if we consider a vehicle capacity constraint, given the fact that private vehicles usually have a few seats. Thus we introduce the slugging problem with the capacity constraint.

As mentioned in Sec. 4.3.1, a trip can be associated with multiple travelers who ride together. In other words, these travelers have the same origin, destination, and start time. Therefore, each passenger trip in the graph should be tagged with a label which represents the number of travelers associated with the trip. We do so as follows.

Definition 5 A slugging graph $G = (V, E)$ is called no-of-travelers-labelled if each node $T_i \in V$ that represents a passenger trip (i.e. has outgoing edges) is associated with a number $T_i.s$, referred to as the size of node T_i.

Each driver trip also has a number of seats available for passengers. In other words, each driver trip is associated with a number of travelers. However, it may still have available seats in the car to take slugging passengers. This availability is represented in the slugging graph as follows.

Definition 6 A slugging graph G=(V, E) is called no-of-available-seats labelled if each node $T_i \in V$ that represents a driver trip (i.e. has incoming edges) is associated with a label $T_i.c$, referred to as the capacity of node T_i.

Definition 7 *A slugging plan $G_S = (V, E_S)$ is capacitated if each driver trip in G_S takes at most $T_j.c$ additional passengers, i.e. $\forall T_j \in V, \Sigma_{(T_i,T_j)\in E_S} T_i.s \leq T_j.c$.*

Now we define the *Capacitated Slugging Problem* as follows.

Problem 4.3 Given a slugging graph $G = (V, E)$ that is no-of-travelers-labeled, no-of-available-seats-labeled and benefit-labeled, and a number $R \in \mathbb{R}^+$, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a capacitated slugging plan with the benefit at least R. We refer to this as the Capacitated Slugging Problem (CSP).

Theorem 4.3 *CSP is NP-Complete.*

Proof: First, it is easy to see *CSP* is in NP. That is, given a subgraph of G, denoted by G_S, we can verify whether G_S is a -capacitated slugging plan and if so, whether its benefit is at least R. Now, we prove *CSP* is NP-hard by reducing the *0/1 Knapsack Problem* to *CSP*.

The *0/1 Knapsack Problem* is known to be NP-hard [46]. The decision version of the problem is defined as follows: given a set of n items, $\{p_1, p_2, \dots, p_n\}$ and a knapsack of capacity W. Each item

p_i has a size w_i and a value v_i. The question is whether or not we can pack items worth at least R into the knapsack without exceeding its capacity and without splitting items.

Given an instance of *0/1 Knapsack Problem*, we can build an instance of *CSP* as follows. First we construct a slugging graph $G = (V, E)$ as follows. Let the node set $V = \{T_{p_1}, T_{p_2}, \dots T_{p_n}, T_d\}$. Construct the edge set E as follows. For each node T_{p_i}, $i = 1,2, \dots n$, we add an edge (T_{p_i}, T_d) to E. Note that the edge set $E = \{(T_{p_i}, T_d)\}$ is indeed transitive and acyclic. Therefore, G is a slugging graph. Next we label the nodes of G. Each node T_{p_i} is labeled with a size equals to w_i. The capacity of node T_{p_i} does not matter since they can only be passenger trips. Node T_d is labeled with a size equals to 1 and a capacity equals to $W + 1$. Next we label the edges of G with a benefit. Each edge (T_{p_i}, T_d) is label with a benefit $B(T_{p_i}, T_d)$ equals to v_i, for $i = 1,2, \dots n$. Now G is slugging graph that is no-of-travelers-labeled, no-of-available-seats-labeled and benefit-labeled.

It can readily be shown that the constructed instance of *CSP* has a capacitated slugging plan with a benefit of R if and only if the instance of *0/1 Knapsack Problem* can pack items worth at least R into the knapsack.□

4.3.3.1 A special case of CSP

A special case of *CSP* where the capacity of each car is 2, and all trips are associated with only one traveler, is polynomial-time solvable. We prove it formally as follows.

Problem 4.4 *Given a slugging graph $G = (V, E)$ that is no-of-travelers-labeled where the size of each passenger node is 1, and no-of-available-seats-labeled where the capacity of each driver node is 1, and benefit-labeled, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a capacitated slugging plan with the maximum benefit. We refer to this as the 1-traveler-1-availability Capacitated Slugging Problem (1t1CSP).*

Theorem 4.4 *The 1t1CSP is solvable in $O(|V||E|\log|V|)$ time.*

Proof We will show that the *1t1CSP* is equivalent to the maximum weighted matching problem. A matching of a graph is a set of pairwise vertex-disjoint edges. The maximum weighted matching problem is defined as: given an edge-weighted undirected graph $G_M = (V_M, E_M)$, find the matching where the sum of the weight of the edges in it is maximum.

Given an slugging graph $G = (\mathrm{V},\ E)$ of the *1t1CSP*, we construct a weighted undirected graph G_M as follows: $V_M = \mathrm{V}$, $E_M = E$, the weight of an edge $e \in E_M$ equals to the benefit of e. Since E is acyclic, G_M contains no self-loops. Thus, each matching M of G_M is a legitimate capacitated slugging plan and the sum of the weight of edges in M equals to the benefit of the slugging plan.

Since the maximum weighted matching problem is solvable in polynomial time [45], we can also solve the *1t1CSP* in polynomial time using the same algorithm. The running time of this algorithm is $O(|V||E|\log|V|)$. □

4.3.4Delay-Bounded Slugging

In addition to the vehicle capacity constraint, it is also natural to constrain *SP* by a bounded travel time delay. As mentioned in Sec 4.3.1, in the pre-processing stage (that uses a map), for each mergeable pair (T_i, T_j), we compute the travel time delay for the passenger trip T_i, denoted by $\Delta_{i\to j}$. Intuitively, $\Delta_{i\to j}$ is the delay incurred by T_i due to the fact that T_i needs to walk to/from T_j's origin/destination, and possibly wait for T_j to start. More specifically, the delay equals to the difference: (the arrival time of T_i when the passengers ride with T_j (i.e. walk to/from origin/destination of T_j)) – (the arrival time of T_i when the passengers ride in their own vehicle from their origin directly to their destination). Now we define the travel time delay representation in the graph.

Definition 8 A slugging graph $G = (V, E)$ is called delay-labelled if each edge $(T_i, T_j) \in E$ is associated with a label $\Delta_{i\to j}$, which represents the travel time delay of T_i with respect to (T_i, T_j).

The travelers of each trip T_i can specify a threshold which represents their maximum tolerable travel time delay. We define the delay threshold representation in the graph.

Definition 9 A slugging graph $G = (V, E)$ is called delay-threshold-labelled if each node $T_i \in V$ is associated with a label $T_i.\delta$, referred to as the delay threshold of node T_i.

The travel time delay constraint means that all edges outgoing of a node with a travel time delay exceeding the delay threshold of the node need to be filtered out. Thus we define a delay-bounded-slugging-graph that satisfies this property.

Definition 10 *Given a slugging graph $G = (V, E)$ that is delay-labeled and delay-threshold labeled, the delay-bounded slugging graph $G_\delta = (V, E_\delta), E_\delta \subseteq E$, is a subgraph of G where $\forall (T_i, T_j) \in E_\delta$, $\Delta_{i \to j} \leq T_i.\delta$.*

Now we introduce the *delay-bounded slugging problem.*

Problem 4.5 *Given the delay-bounded slugging graph G_δ that is benefit-labeled, and a number $R \in \mathbb{R}^+$, find a subgraph G_S of G_δ that is a slugging plan with a benefit of at least R.*

4.3.5Delay Bounded and Capacitated Slugging and Its Heuristics

In practice, both the capacity constraint and the travel time delay threshold constraint are important. Thus, we combine them to form the following problem.

Problem 4.6 *Given the delay-bounded slugging graph $G_\delta = (V, E_\delta)$ that is no-of-travelers-labeled, no-of-available-seats-labeled, and benefit-labeled, and a number $R \in \mathbb{R}^+$, find a subgraph $G_S = (V, E_S), E_S \subseteq E_\delta$ that is a capacitated slugging plan with a benefit of at least R. We refer to this as the Delay Bounded and Capacitated Slugging Problem (DBCSP).*

Theorem 4.5 *The DBCSP is NP-Complete.*

Proof Obvious, since *DBCSP* is a generalization of *CSP*. □

Since the *DBCSP* is NP-Complete, we propose two greedy heuristics for the *DBCSP,* namely *Greedy-Benefit* and *Greedy-AVG-Benefit*. Both heuristics work in an iterative way. That is, each heuristic greedily chooses one driver trip T_d based on certain criteria. Intuitively, *Greedy-Benefit* chooses the driver trip that collects the maximum benefit of its incoming edges, and *Greedy-AVG-Benefit* chooses the driver trip that collects the maximum average benefit of its incoming edges.

To compute the maximum benefit and the maximum average benefit, we need to solve an instance of the 0/1 knapsack problem for each driver trip. Each driver trip T_d and all its passenger trips T_p where $(T_p, T_d) \in E_\delta$, form an instance of the 0/1 knapsack problem (see proof of Theorem 2). That is, trip T_d is considered the knapsack with a capacity equals to $T_d.c$ and each T_p is considered an item with a value equal to $B(T_p, T_d)$ and a size equal to $T_p.s$.

Since the 0/1 knapsack program is NP-complete, we employ an approximation algorithm called *Efficiency Greedy (EG)* approximation algorithm [46]. The *EG* algorithm outputs the larger between the following two numbers: (i) the total value when packing items into the knapsack in non-increasing order of their efficiencies (i.e. the ratio of value to size); (ii) the value of the single item which is most valuable among all items. It is known that the *EG* algorithm has a worst-case performance bound of 2 [46].

Intuitively, at each iteration the *Greedy-Benefit* heuristic applies the *EG* algorithm to each driver trip, and selects the driver trip with the maximum benefit computed by *EG*. This trip and its passengers are eliminated from the slugging graph, and then the next iteration is started. The *Greedy-AVG-Benefit* is identical, except that the driver trip selected is the one with the (maximum benefit / number of passenger trips selected by *EG*).

Algorithm 2: Heuristics Algorithms of $DBCSP$

Data: the delay-bounded slugging graph $G_\delta = (V, E_\delta)$ that is no-of-travelers-labeled, no-of-available-seats-labeled and benefit-labeled
Result: a ridesharing plan G_S

1 $G_S \longleftarrow \emptyset$
2 **while** $E_\delta \neq \emptyset$ **do**
  `/* calculate driver trips */`
3 $driverTrips \longleftarrow \emptyset$
4 **for** *edge* $(T_i, T_j) \in E_\delta$ **do**
5 $driverTrips \longleftarrow driverTrips \cup \{T_j\}$
  `/* select a driver trip */`
6 **for** $T \in driverTrips$ **do**
7 Calculate $B_{appr}(T)$ using the EG algorithm
8 pick driver trip T_d with the maximum $B_{appr}(T)$ `/* GREEDY-BENEFIT heuristic */`
  `/* replace` $B_{appr}(T)$ `with` $B_{appr}(T)/n$ `when using the GREEDY-AVG-BENEFIT heuristic */`
9 $S_p \longleftarrow \{T_p | T_p$ is a passenger trip such that $(T_p, T_d) \in E_\delta$ and selected by the EG algorithm $\}$
  `/* update ridesharing plan` G_S `*/`
10 **for** $T_p \in V_\delta$ **do**
11 add (T_p, T_d) to G_S
  `/* update the` E_δ `*/`
12 $S_{chosen} \longleftarrow \{T_d\} \cup S_p$
13 **for** *trip* $T \in S_{chosen}$ **do**
14 remove T and all its incident edges from graph G_δ;
15 return G_S

Figure 28 Heuristics for the DBCSP

More precisely, denote by $B_{appr}(T_d)$ the result of the instance of the 0/1 knapsack program formed for trip T_d output by the *EG* algorithm. Denote by n the number of passenger trips that are selected for driver trip T_d by the *EG* algorithm. Then, in each iteration, the *Greedy-Benefit* and *Greedy-AVG-Benefit* heuristic select the driver trip with the maximum $B_{appr}(T_d)$ and $B_{appr}(T_d)/n$, respectively. Once a driver trip T_d is picked, the set of passenger trips that are merged into T_d are also determined by the *EG* algorithm. Next the delay-bounded slugging graph is updated by removing the nodes of the driver and its passengers, and the edges that touch upon them. This update completes an iteration, and a new iteration then starts. The algorithm terminates when the slugging graph becomes empty.

Figure 28 summarizes the algorithm for the proposed heuristics. Lines 6~7 calculates $B_{appr}(T_d)$ for every driver trip T_d in an iteration. Since there are $O(|V|)$ driver trips in an iteration, and the computation of $B_{appr}(T_d)$ for each T_d by the *EG* algorithm runs in $O(|V|log|V|)$, therefore, the

selection of the driver trip in an iteration runs in $O(|V|^2 \log |V|)$. The updating process takes $O(|V|)$ for each selected trip and takes $O(|V|^2)$ in total, as there are at most $|V|$ trips selected in an iteration. Since there are $O(|V|)$ iterations, the time complexity of the greedy heuristics is $O(|V|^3 \log |V|)$.

To illustrate the *Greedy-Benefit* and *Greedy-AVG-Benefit* heuristics, please consider the delay-bounded slugging graph shown in Figure 29 (a). The number on each edge represents its benefit. Assume that the capacity and the size of each node is 4 and 1, respectively. The optimal slugging plan for this simple example is shown in Figure 29 (b), with a benefit of 38.

(a) a delay-bounded slugging graph (b) the corresponding optimal slugging plan

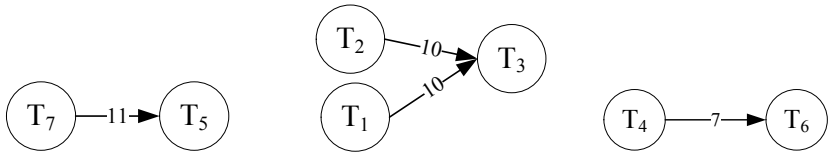
Figure 29 An example of delay-bounded slugging graph

Now we generate a slugging plan using the greedy-based heuristics. Let us first consider the *Greedy-Benefit* heuristic. In the first iteration, T_3 is chosen as the driver trip because the maximum benefit that it can collect from its incoming edges is the largest among all driver trips. Then the graph is updated by deleting all edges associating with any of T_1, T_2, T_3. In the next iteration T_6 is chosen as the driver trip. The graph then updates again and becomes empty of edges. The edges selected in each step are shown in Figure 30 and the benefit of the resulting slugging plan is 33.

(a) edges selected in 1st step (b) edges selected in 2nd step

Figure 30 Running example of the Greedy-Benefit heuristic

In contrast, *Greedy-AVG-Benefit* chooses T_5 in the first iteration because T_5 has the largest average benefit of passenger trips. Then T_3 is selected in the second iteration and T_6 is selected in the third iteration. Figure 32 shows edges selected in each iteration and the final slugging plan has a total benefit of 38.



(a) 1st step (b) 2nd step (c) 3rd step

Figure 31 Running example of Greedy-AVG-Benefit heuristic

For the example shown in Figure 31, *Greedy-AVG-Benefit* is coincidentally optimal. But the greedy heuristics cannot always guarantee the optimal solution. For example, neither heuristics is optimal for the example shown in Figure 32, assuming that the size of each node is 1.

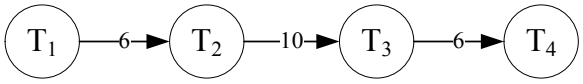


Figure 32 An example for which heuristics are sub-optimal

4.3.6Dynamic Slugging

The basic and constrained slugging problems that we have discussed are presented in a static context where all trips are known before the calculation of the slugging plan and the slugging plan is only calculated once. In this section, we discuss how to deal with the slugging problem in a dynamic context where the computation of a slugging plan is performed many times on the fly as the announcements of trips are continuously arriving.

Figure 33 illustrates an instance of the dynamic slugging problem that involves five trips. The announcement of each trip is depicted by a circle and the start time of each trip is depicted by a diamond. On the one hand, it is necessary that there exists a temporal gap between the

announcement and the start time for each trip; otherwise (i.e. if trips start at the same time when they are announced) there will be no room for ridesharing. On the other hand, such a temporal gap may be small (a few minutes) since these trips are dynamically generated. In the extreme case where these temporal gaps are huge (e.g. hours or even a day), the dynamic problem then degenerates to the static problem. Here we assume that the temporal gap between the announcement and trip start time is the same for all trips and denote this number by G. In other words, each trip is announced G time units before its start-time.

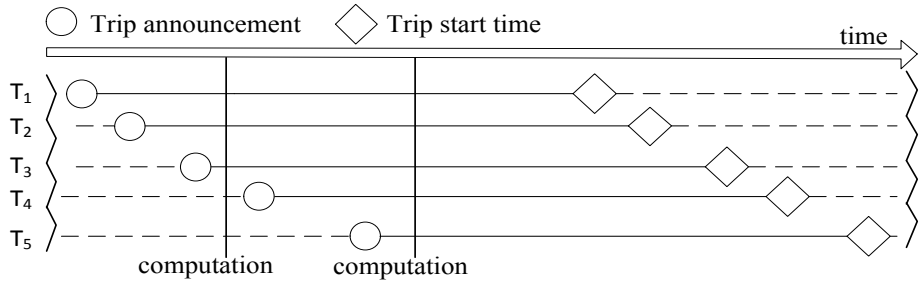


Figure 33 The dynamic slugging problem

As in the static case, the objective of the dynamic problem is maximizing the total benefit. In the dynamic slugging problem, the slugging plan is computed and executed every f seconds as depicted by the vertical lines in Figure 33, where f is referred to as the *decision interval*. Now we describe how the overall benefit of ridesharing is calculated for the dynamic problem. Once a trip is announced, it remains in the input set of the slugging plan computation until either of the following events happens: (i) the trip is included in the slugging plan as a result of a computation; (ii) the start time of the trip is reached (i.e. the trip starts without any ridesharing). The aggregate slugging plan (of all trips) is simply the union of all slugging plans calculated in each decision time point, and thus the overall benefit is computed based on the aggregate plan.

Table 6 An example of the dynamic slugging problem

	The Set of Trips As the Input	Calculated Slugging plan	Benefit
First computation	$\{T_1, T_2, T_3\}$	$\{(T_1, T_2)\}$	1

Second computation	$\{T_3, T_4, T_5\}$	$\{(T_3, T_5), (T_4, T_5)\}$	2
Aggregate	$\{T_1, T_2, T_3, T_4, T_5\}$	$\{(T_1, T_2), (T_3, T_5), (T_4, T_5)\}$	3

Table 6 gives a running example of how the benefit is computed in the dynamic context for the example given in Figure 33. Suppose that the first computation outputs a slugging plan $\{(T_1, T_2)\}$. Since T_3 is not included in the plan and it has not reach its start time, T_3 remains in the input set. Suppose that the second slugging plan computation yields $\{(T_3, T_5), (T_4, T_5)\}$. Therefore, the aggregate slugging plan is $\{(T_1, T_2), (T_3, T_5), (T_4, T_5)\}$ and the aggregate benefit is 3, assuming that the benefit of each merging is 1.

The value of f should be tuned carefully in order to maximize the benefit of ridesharing. We evaluate the optimal value of f experimentally and present the results in the next section.

4.4Evaluation

4.4.1Setting

We conducted experiments using a taxi GPS trajectory data set [70]. The dataset contains real traces from more than five thousand taxis in Shanghai during a single day. These taxis have been equipped with GPS receivers (one for each). The GPS receivers periodically report their current states to a data centre via GPRS links. Each record has a format <*TAXI_ID, TIMESTAMP, LONGITUDE, LATITUDE, OCCUPIED*>. Intuitively, each sequence of consecutive records where the *OCCUPIED* field constantly equals to 1 is an occupied trip. Figure 34 shows a *TAXI_ID* and *TIMESTAMP* ordered snippet of a GPS trajectory data file and the blue rectangle represents an occupied trip.

```
105,2007-02-20 17:05:42,121.532800,31.231500,0
105,2007-02-20 17:06:02,121.532100,31.231100,1
105,2007-02-20 17:07:02,121.534100,31.225100,1
105,2007-02-20 17:11:06,121.537000,31.214300,1
105,2007-02-20 17:12:07,121.545800,31.212600,1
105,2007-02-20 17:14:09,121.551500,31.206100,1
105,2007-02-20 17:15:10,121.551500,31.206100,1
105,2007-02-20 17:16:11,121.551500,31.205800,1
105,2007-02-20 17:17:12,121.554800,31.198000,1
105,2007-02-20 17:17:45,121.556500,31.197800,0
```

Figure 34 A snippet of taxi trajectory data that defines a trip

For our experiments, each such occupied trip defines a trip T as follows: the time stamp and the GPS point of the first record in the sequence defines the start time and origin of T, respectively; the time stamp and the GPS point of the last record in the sequence defines the end time and destination of T, respectively; the travel time of T then equals to the start time minus the end time; the travel distance is the road network distance between the origin and destination as obtained via the Google Map API. Out of 60 thousand occupied trips extracted from the data set, we selected 39 thousand trips which last over 5 minutes. This constituted our experimental pool of trips. The average travel time and travel distance of these trips is 12.3 minutes and 6.3 kilometres, respectively.

We evaluate the *DBCSP* in all experiments. The benefit of ridesharing is measured by the saving in vehicle travel distance. To compute the edge set of the slugging graph, we assume that all travelers walk at the same speed, denoted by W, and always walk along the shortest road path between two locations.

Table 7 Parameter setting in the experiments

Notation	Definition	Default Value
W	travelers' walking speed	5 km/h
δ	travel time delay threshold	20 minute
C	vehicle capacity	3
G	temporal gap between the announcement and trip start time	15 minute

In all the experiments, we assume that each trip is associated with only one traveler and she is willing to be either a passenger or driver in the slugging plan. For simplicity, we assume that all

trips have the same travel time delay threshold, denoted by δ; and all cars have the same number of seats, denoted by C. Table 7 lists default values for the parameters used in our experiments.

4.4.2Upper Bound on the DBCSP

Algorithm 3: Calculate an upper bound on the benefit of the ridesharing plan for the *DBCSP*

Data: the delay-bounded slugging graph $G_\delta = (V, E_\delta)$ that is no-of-travelers labeled, no-of-available-seats labeled and benefit-labeled
Result: an upper bound B_{upp} on the benefit of a ridesharing plan

```
1  passengerTrips ⟵ ∅
2  driverTrips ⟵ ∅
3  for edge (T_i, T_j) ∈ E_δ do
4      passengerTrips ⟵ passengerTrips ∪ {T_i}
5      driverTrips ⟵ driverTrips ∪ {T_j}
6  sum_1 ⟵ 0
7  sum_2 ⟵ 0
8  for trip T_i ∈ V do
9      if T_i ∈ passengerTrips then
           /* the upper bound if all passenger trips are merged */
10         sum_1 = sum_1 + B(T_i, T_d) , d is any number such that (T_i, T_d) ∈
           E_δ
11     if T_i ∈ driverTrips then
           /* the upper bound if all driver trips collect maximum
              benefit                                              */
12         L ⟵ top C − 1 passenger trips of the driver trip T_i
13         for trip T_p ∈ L do
14             sum_2 = sum_2 + B(T_p, T_i)
15 B_upp ⟵ min(sum_1, sum_2)
16 return B_upp
```

Figure 35 An upper bound of the DBCSP

What is the maximum benefit that can be obtained by slugging? To answer this question, we obtain an upper bound on the benefit of a slugging plan for the *DBCSP* by relaxing either one of the two constraints imposed by the definition of slugging plan (see Def. 2). Relaxing Condition (ii) and the capacity constraint, we get an upper bound, denoted by B^1_{upp}, by merging each passenger trip T_i into some driver trip T_j regardless whether or not T_j has been merged into some other trip and regardless whether or not T_j has any available seat left. Relaxing Condition (i), we get another upper bound, denoted by B^2_{upp}, by making each driver trip T_d collect the maximum benefit regardless whether or not any its passenger trip T_p has been merged into any driver trip other than T_d. The smaller value of B^1_{upp} and B^2_{upp} is used as the final upper bound. Figure 35 shows the

algorithm that outputs this bound. It is easy to see that the time complexity of the algorithm is $O(|V|^2)$.

4.4.3DBCSP With Varying Travel Delay

First we evaluate the proposed greedy-based heuristics by fixing the vehicle capacity and varying the travel time delay threshold. Experiments are performed for various thresholds of travel time delay $\delta \in [5,\ 20]$ minutes with an increment of 5 minutes.

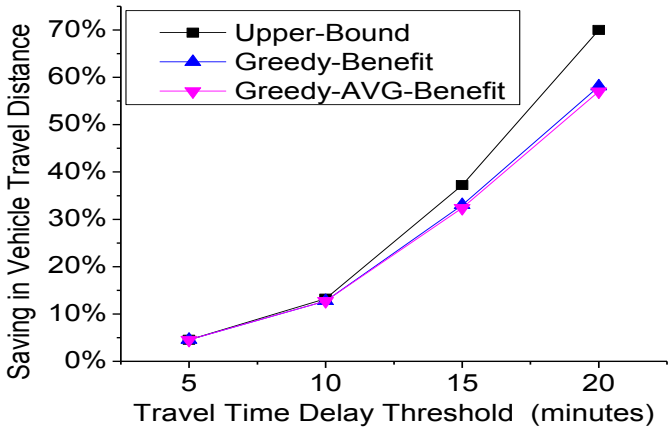
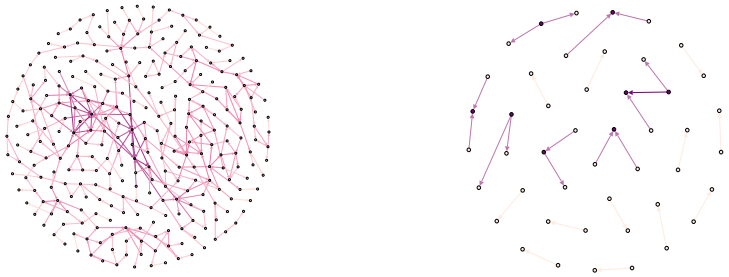


Figure 36 DBCSP with varying delay thresholds

As Figure 36 shows, when the travel time delay threshold is large, both the *Greedy-Benefit* and the *Greedy-AVG-Benefit* perform consistently close to the upper bound. When $\delta = 20$, these heuristics have a 59% saving while the upper bound is 70%. Given the average distance of these trips is 6.3 kilometers and the size of our data set is 39 thousand, the 59% saving in vehicle travel distance is 144, 963 kilometers which means the reduction of over 4.5 thousand gallons of gasoline and 71 tons of carbon dioxide emission.

When the travel time delay threshold δ is small, there is no significant difference between the greedy-based heuristics and the upper bound. This is because, with a small δ, slugging opportunities are so rare that the slugging graph is extremely sparse. As a result, the graph admits very few possible slugging plans.



(a) δ= 20 minutes (b) δ = 5 minutes

Figure 37 Visualization of a delay-bounded slugging graph

For example, Figure 37 (a) and (b) visualize the delay-bounded slugging graph of a subset of trips when δ is 20 and 5 minutes, respectively. The darker the node's color is, the larger the node's in-degree (i.e. the number of incoming edges) is. When δ is 20 minutes, the graph is weakly connected. When δ is 5 minutes, most edges disappear and the graph is scattered into many disconnected components, each of which comprises of at most four nodes. In this case, it is clear that different algorithms will make little difference in the resulting slugging plan.

4.4.4DBCSP with Varying Vehicle Capacity

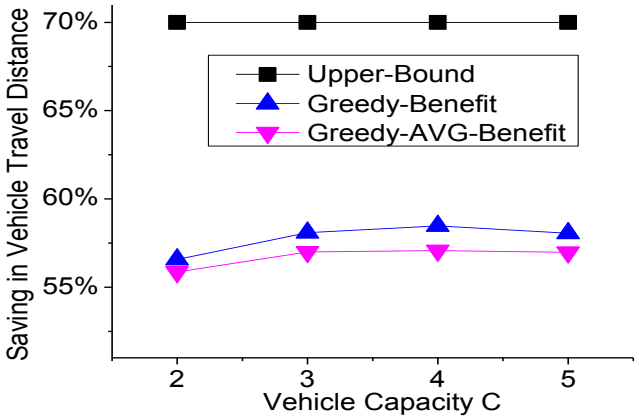


Figure 38 DBCSP with varying vehicle capacities

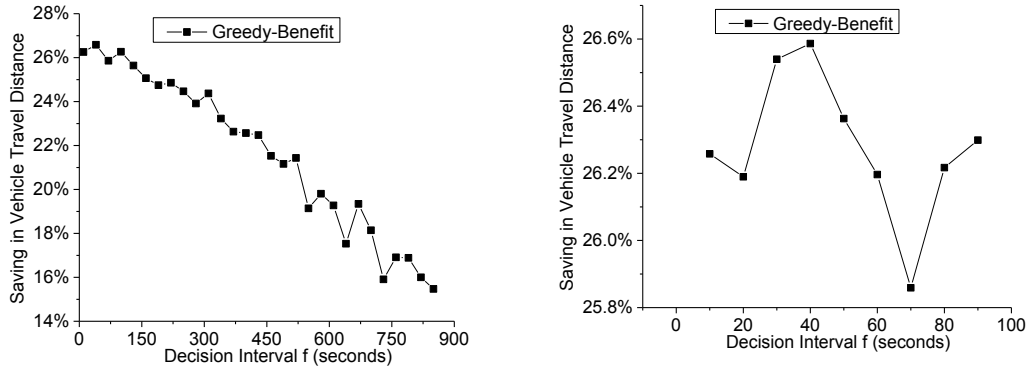
In this experiment, the vehicle capacity varies, i.e. $C \in [2, 5]$ while the travel time delay threshold is fixed. Figure 38 shows the performance of different heuristics. The result is consistent

to that of Figure 36 , i.e., the greedy heuristics perform relatively close to the upper bound consistently. In addition, it is shown that the saving percentage saturates as the capacity increases, given the travel time delay is bounded. This is because the average number of incoming edges for each driver trip in the input graph is small, which can also be observed from Figure 37. Even when $\delta = 20$ minutes, as shown in Figure 37 (a), most driver trips have only one or two passenger trips that can be merged into them, and the average number of passenger trips for a driver trip is 1.38.

4.4.5Dynamic DBCSP

In this experiment, we evaluate the *Greedy-Benefit* heuristic in a dynamic context. C is set to be 3 and δ is set to be 15 minutes. We set f to be smaller than G, otherwise many trips will start without encountering any computation of a slugging plan.

Since G is 15 minutes, we set the range of f to [10, 880] second with an increment of 30 seconds. Figure 39 (a) shows a clear trend of decrease in benefit as the value of f increases. But the figure also clearly shows that the benefit fluctuates locally. To see the fluctuation more clearly, we further fine tune the value of f within a relative small range. Figure 39 (b) shows the benefit fluctuating as decision interval f increase from 10 seconds to 100 seconds with an increment of 10 seconds. The saving rate reaches the maximum when f equals to 40 seconds.



(a) the decrease trending (b) fine tune of f

Figure 39 Impact of the decision interval

Figure 39 can be explained by two conflicting factors. On the one hand, as f increases, the computation of slugging plans becomes less frequent, so travelers of passenger trips cannot start walking until the time of the next computation. Therefore, the wasted time costs many ridesharing opportunities and thus decreases the benefit. On the other hand, as f increases, the input pool of trips for each computation of the slugging plan becomes larger and thus the benefit may increase. Figure 39 (a) suggests that the first factor wins the tug-of-war, so we have an overall decreasing trend with local fluctuations. It is also revealed from Figure 39 that the saving rate at its peak (i.e. f=40 seconds) is about 26.6%. In contrast, the saving rate of the corresponding static problem with the same capacity and delay parameters (i.e. C=3 and $\delta = 15$ minutes) is 33%.

4.5Discussion

We have analysed slugging, an increasingly popular form of ridesharing, probably due to its simplicity. Specifically, we have formalized and studied the slugging problem. For the unconstrained slugging problem, we have proposed a quadratic algorithm to solve it optimally. We prove the NP-completeness of its constrained variants. For the constrained variant, we proposed heuristics and evaluated them on a data set consisting of tens of thousands of real trajectories of taxi cabs in Shanghai. The heuristics achieved near-optimal travel distance savings. The experimental results suggest that the saving in travel distance can reach as much as 59% whereas the optimal slugging plan achieves at most 70% savings. Given the size of our data set is 39K and the average distance of trips in the data set is 6.3 kilometres, the saving equals to 144,963 kilometres, which means the reduction of over 4.5 thousand gallons of gasoline and 71 tons of carbon dioxide emission. In addition, for the dynamic slugging problem, we evaluated the impact of the decision interval, i.e. how frequently to run the slugging algorithm, on the overall benefit.

In the future, we are going to refine this work towards a working system by considering and modelling other practical individual preferences such as riders' social preferences [50]. For the dynamic slugging problem, we are going to improve the plan calculation algorithm by considering

a probabilistic model which looks ahead, i.e. predicts the future announcement of trips. The objective function here optimized the overall benefit. It is possible to consider individual trips and minimize the walking distance and/or travel time delay of an average passenger trip. These extensions will be considered in future work.

Chapter 5

Volunteer Transportation Information System

5.1 Introduction

User Generated Content (UGC), i.e., users voluntarily providing information, has the promise to revolutionize information in transportation. Traditionally, information on traffic conditions (congestion, incidents and so on) has been available from road sensors such as inductive loop detectors that are embedded in the pavement of highways or CCTV's above roads or in sensor systems that are increasingly being deployed on transit vehicles such as trains and buses. The number of sensors in a car has also rapidly increased, beyond traditional in-vehicle sensors, to those that give automobiles situational awareness.

With the proliferation of smartphones, cell phones and other mobile devices, transportation users have a unique opportunity to become a part of the transportation sensor network. Such UGC may arise from two types of sensing activities: 1) opportunistic sensing where, for example, individual drivers, pedestrians, transit users or bicyclists passively volunteer their location and time at a location by means of sensors in mobile devices that they are carrying; or 2) participatory sensing, where information is actively volunteered, for example, by entering information by text or voice, about events such as incidents or delays that they have encountered while traveling.

Either way, such sensing systems provide the opportunity to generate unique content about the state of the transportation system. This is important because relevant information is often generated by regular travelers in a mobile environment. Mobile social networks, where individuals with similar interests share information with one another using the mobile phone, have been studied in the literature (see e.g., [58, 82]), mobile social networks targeted to transportation (which we will

call mobile communities for transportation) have been the subject of little to no attention to date. Examples of mobile communities for transportation include the following:

1) Mobile communities for travel information: In such communities, participants publish and subscribe to information on travel congestion, delays, weather effects, and incidents such as accidents, public transportation delays, detours, service disruptions due to repair and maintenance work in train and bus stations and stops, and other factors.
2) Mobile communities integrating health and fitness concerns with transportation: In such communities, participants may share with subscribers information on opportunities to bike instead of drive, or to alight from a bus a few stops away in a particular area so that they can walk to their final destination, and also connect a user to biking or walking partners in such situations so that daily fitness goals can be met within a community environment.
3) Mobile communities for safe mobility: A major deterrent to non-motorized or public transportation use in urban areas is personal safety concerns. This type of mobile community for transportation may be targeted to publishing information on safe use of multimodal transportation systems, ranging from sharing safe routes between two points to bike or walk or to find a walking buddy in real-time from a transit station or stop. Parents in a neighborhood may be a part of a mobile social network to share information on the availability of an adult to walk with a group of school-children from home to a public transportation station in a high-crime neighborhood or to a neighborhood park, when crossing unsafe streets is required along the way.
4) Mobile communities for shared resource transportation: Mobile communities for transportation may also form around demand-responsive transportation systems such as dynamic ride-sharing or even station car sharing (where cars for hire may be used from a transit station to an employment location that is otherwise not accessible). Opportunities may be available to share information on bike-sharing and car-sharing as well.

We propose to study several social and technical issues associated with mobile transportation communities, with the ultimate goal of improved system design. As such systems grow in complexity, interconnectedness, heterogeneous users with myriad socio-demographic and geographic distribution, we may increasingly face unexpected emergent behaviors that are not easily predictable from the behavior of individual publishers/subscribers. As proof of concept, we will develop a technology, called the Volunteered Transportation Information System (VTIS), which will serve as a basis to understand the social and technical issues to be investigated. The proposal lays out the major social and technical issues to be investigated and the elements of the VTIS to be used as part of the study.

The social aspects of building and sustaining such communities pose several research questions. How can users be recruited? How would they learn about the system, how to use it, and what its benefits would be? How would interactions among agents evolve over time? Would a certain level of users be necessary before the benefits of the systems become evident? Are there unanticipated consequences such as the breakdown of such systems during large-scale emergencies or evacuation situations? What are the fundamental limits, barriers, and saturation levels that may set in with "information overload" with these systems? These questions point to the fact that studying mobile communities for transportation may lead to insights about emergent intelligence as a property of a group of agents rather than of individual constituent agents, whereby intelligent behavior emerges from the interaction between agents.

A major distinguishing feature of mobile communities for transportation are the speeds at which publications and subscriptions need to work, given the speeds at which transportation systems work. Observe that much of the information may decay to the extent that it is no longer valid, useful or reliable. Speeds also have implications on Human-Computer Interaction aspects of the technologies supporting such mobility communities since the driving or travel environment may pose difficulties to publish information, compared to if the person is inside a building or in a stationary situation.

Observe however, that publications may be entered by speaking into smartphones equipped with speech-to-text software such as Siri.

Concerns of trust management take on a heightened tone, as the prospects of personal harm and safety risks can be substantial, if the information is wrong or malicious. Questions of incentives can also be unique in such situations because the benefit to the subscriber is not always obvious (for example, what would motivate a person to report a delay, which would benefit others and not oneself, since the person is already stuck in traffic)?

The approach taken in this proposal, to use mobile communities for transportation as a case study to study socially intelligent computing has three benefits. First, research into the social and technical aspects of such communities advances the state of research in online and mobile communities and identifies how emergent behaviors, interactions among agents, reaching critical mass, and related properties, may be able to inform the design of future systems. Second, research into interactions among users and with the technologies supporting mobile transportation communities will advance mobile technologies, especially where higher speeds, distractions, and issues of user cognition in complex real-world environments are concerned. Third, fundamental research into user generated content by mobile transportation communities will lead to computationally efficient methods to retrieve useful intelligence from unprecedented amounts of greatly heterogeneous information streams.

Social media is among one of today's most powerful methods of rapidly spreading news and key information to a large population ([54]). In fact, half of all U.S. adults are now on social networks such as Facebook and Twitter. This implies a tremendous potential reachability when using social media to share information. In this project we will build our proof concept prototype (i.e., VTIS) upon existing social media. VTIS is dedicated for the sharing of real-time traveler information such as traffic conditions, accidents, road maintenance, parking spaces availability, special events, weather, etc. VTIS will exploit the capability of social media to scale to millions of users and constant streams of publications/subscriptions. This scalability would be expensive to

reach if the system was to be built from scratch. Yet there are several research issues that need to be solved when using social media to share traveler information, which we will study in this project. These research issues include:

1) **Semantic structuring of social networks.** Find proper mapping between the organization of social networks and the spatio-temporal domains that are of interest to travelers.
2) **Trust management.** How to distinguish between truthful and false publications? The trust management problem remains an open issue for Intelligent Transportation System (ITS) applications as well as traditional on-line e-business applications such as eBay and Amazon. We will investigate and review existing work on dealing with data trust issues and adapt and apply available techniques to VTIS.
3) **Publication Ranking.** Travelers are presumably on the move all the time. For the sake of safety, it is desirable to minimize the interactions required between the traveler and VTIS. One way to minimize interactions is to rank publications such that the most relevant ones are presented to a user for viewing, commenting, or editing.
4) **Incentive Mechanisms.** We will study incentive mechanisms to stimulate users to participate in the social networks.

In general, we believe that social media has potential to serve as a platform for building mobile collaborative communities. In addition to transportation, mobile collaborative communities can be formed in other application domains. For example, the customers in a shopping center may form a community to share coupon information, or the attendees in a conference may form a community to find the match making of expertise and interests, and so on. The technologies that we will develop in this project, including building, sustaining, and studying the evolving growth dynamics and understanding emergent intelligence in mobile communities for transportation will lend approaches to general mobile collaborative communities.

5.2 Related Work

5.2.1 Publish/Subscribe

As a communication paradigm, publish/subscribe mode has been studied well in both mobile P2P environment and Internet-based environment. [81] proposes a publish/subscribe implementation for MANETs. In the implementation, subscriptions are only deployed locally due to the fact that the paths utilized by the subscription forwarding strategy in server overlay networks quickly become stale in a mobile environment. When receiving an event from a neighbor, a mobile node matches the event to its own subscriptions and broadcasts the event, as long as it is still valid given the current location and time. [56, 98] propose using a tree structure to implement publish/subscribe in MANETs. But their approaches assume that there is only one publisher in the network. Triantafillou and Aekaterinidis [108] present a different approach to support P2P applications via building a pub/sub middleware over a structured P2P network such as Chord [106]. But that solution is restricted to an environment of static peers.

The above references all implement publish/subscribe in a fully distributed manner. Publish/Subscribe in VTI exploits Twitter as the intermediate broker between publishers and subscribers and thus more resembles existing Internet-based publish/subscribe applications, such as RSS, Atom and systems introduced in [43, 103, 128], where all of them put a server between publishers and subscribers. Existing Internet-based publish/subscribe applications generally fall into two categories, namely topic based and content based [57]. In topic based systems, subscribers express interests by simply joining a group defined by a central subject. Whereas content based systems provide much more flexibility in expressing subscribers' interests by allowing them to specify predicates over a set of attributes. As a result, arbitrary queries over the events content can be easily posed by subscribers. However, VTI's publish/subscribe implementation is tightly integrated with Twitter and different from either approach in Internet-based publish/subscribe applications.

5.2.2 Toponym Recognition and Information Extraction

The VTI project is relevant to the field of information extraction, especially to literature on toponym recognition. Toponym recognition is necessary for location based services such as VTI where input data, e.g. publications, may be in an unstructed format. There have been a couple of works [67, 68, 69] on recognizing and resolving toponyms in text.

5.2.3 Route Planer

Route planners are relevant to the project because they provide services which allow subscribers to define routes. Route planners available today generally falls into two categories: form-based and map-based. Many transportation agencies [1, 3, 7] provide web sites that allow users to plan a trip using the public transportation system. They tend to allow the specification of time constraints, mode constraints (some include information for the auto network as well), preferences for walking distance, and how the trip should be optimized (e.g., duration vs. number of transfers). If a valid trip can be constructed the user is presented with an itinerary for its execution. A wide range of algorithms [15, 23, 72] supporting these route queries have been developed to account for the problems with modal transfers, schedules, and cost computation.

The second common class of planning tools has map-based graphical user interfaces [22, 85]. Users may enter their origin and destination via either a form or by clicking points on a map. Unlike most form-based planners, some map-based sites allow for the insertion of multiple stops along the trip and may include some real-time traffic information.

5.2.4 Data Trust in Intelligent Transportation System and Internet

There have been a few works on data trust management in ITS. [29, 38] describe the opinion piggybacking approach, i.e. forwarding peers attaching their trust opinions on the forwarded message, to allow peers to evaluate the trustworthiness of messages via aggregating trustworthiness values provided by previous forwarding peers. Similar to the piggybacking approach, [111] validates the message by considering other peers' opinions on the message. However, here a peer

itself chooses the opinion providers rather than accepting opinions from the forwarding peers. Despite the subtle differences in technique details, these three papers share the high level approach, i.e. validating data based on reputation systems. In contrast, [95] exploit data aggregation to validate received messages. All these existing works consider the data trust issue in a peer-to-peer based environment. In other words, they assume a pure distributed environment with no central server exists and peers have direct interactions between themselves.

In contrast, the VTI application has a client-server architecture where the VTI application serves as the brokers between all travelers. From this perspective, data trust problem in the VTI application has a closer nature to that of applications built around Internet. Existing techniques proposed in centralized applications such as the beta reputation system [60], eBay [97] and Amazon [49] are supposed to be adapted and applied to the VTI application.

5.2.5 Reports Prioritizing

Publication ranking is relevant to the existing work on prioritizing reports for mobile P2P query processing. In [99] the rank of a report is a weighted sum of its popularity, reliability, and size. The paper does not discuss how the weights are determined. In [127] reports are ranked such that the number of replicas of each report is proportional to the square root of its access frequency. According to [32], such a distribution of replicas has the optimal replication performance in minimizing the query cost. In [85] the rank of a report is computed based on its popularity, age, and distance to its producer. In [34, 89, 126] reports are ranked based on an abstract utility function which is to be defined by specific applications.

5.2.6 Incentive Mechanism

Motivation behind volunteer participations in collaborative Web-based efforts such as Open Street Map, Wikipedia, etc., has been studied in the field of Geographic Information System (GIS), and citizen science. Specifically, Coleman introduces the concept of Volunteered Geographic Information (VGI), GIS systems built by volunteer contributors, and analyzes the motivations of

these contributors. [44] study what drives people to volunteer contribute and edit contents on Wikipedia.

Incentive mechanisms have also been studied in the context of mobile ad-hoc networks [24, 131]. These works propose stimulation mechanism for mobile nodes to cooperate in forwarding information to other nodes. Incentive mechanisms have also been studied for static peer-to-peer networks. In this case the static nature of the problem is often relied upon heavily, for example, by "punishing" a user that is found non-cooperative over time.

There have been many efforts in providing incentives done by social networks and location-based services. For example, many applications such as Facebook, Google Latitude, Yelp , etc. provides the so-called "check-in" feature which rewards virtual points to users if they have visited some place of interest recently. These virtual points are often used to exchange or earn "honors" or "titles" defined by the application, e.g. badges and mayorship in Foursquare, moods in Waze [8], etc.

5.3 Architecture

The publish/subscribe (pub/sub) system consists of a set U of users and a set C of accounts. Each account c is associated with a pair (e, t) where t is a period of time and e is an entity such as a bus route, a train station, a road link, an intersection, etc. t is referred to as the temporal coverage of c and e is referred to as the spatial coverage of c. From time to time, each user may send publications to accounts where each publication describes an event that the user observes. An account only accepts publications during its temporal coverage. Each publication pertains to one or more entities and is always sent to the accounts that cover these entities. For example, a publication may report a car accident and thus it is sent to the account that covers the road link where the car accident is observed. For another example, a publication may report the delay of a subway line and thus it is sent to the account that covers the subway line. Each user may

subscribe to and unsubscribe to accounts. A publication that is sent to an account c is disseminated to all the users that are currently subscribed to c.

The temporal coverage is useful when the account is supposed to be active for a certain period of time. For example, we may create a "Halsted, rush-hour" account that is dedicated for publishing the traffic condition of Halsted during rush hours. In general, each account corresponds to a "cube" in the spatio-temporal space, as illustrated in Figure 40.

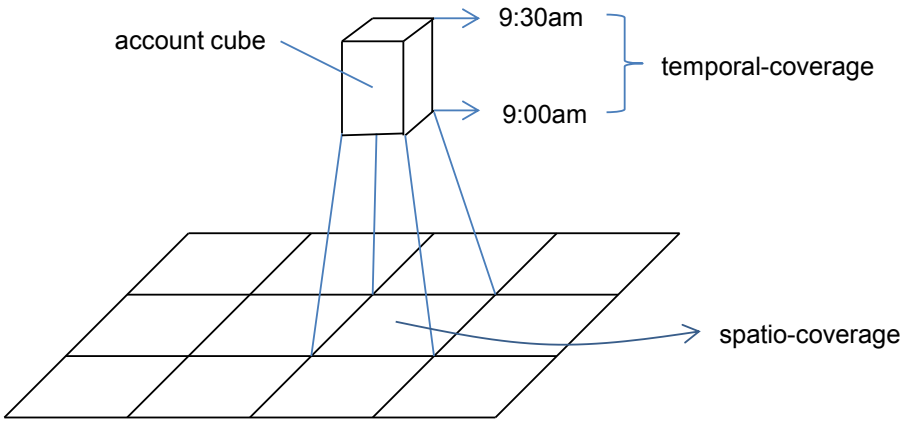


Figure 40 An example of spatio-temporal account

Building the pub/sub system from scratch is difficult and tremendously resource consuming, particularly because the system has to scale to millions of users and accounts. On the other hand, existing social media such as Twitter is already well handling this size of publish/subscribe problem. In this project we will take the advantage of the capability of existing social media and build mobile collaborative communities upon it.

5.4 Implementation

In this section we discuss how we plan to build the VTIS prototype. VTIS is a specialized pub/sub system dedicated for sharing of real-time traveler information among travelers.

5.4.1 Integration with Twitter

As indicated earlier, we intend to exploit the existing social media as the hosting environment for VTIS. Among all the existing popular social media, we have determined that Twitter is the most promising candidate for our purpose. Twitter is an online social networking and micro-blogging service that enables its users to send and read short text-based short posts. An account, i.e. a user, of Twitter can be a person, a group, a system, an application or any meaningful abstraction. Each account is identified by a unique name. Each account A can follow other accounts which are called the *followings* of A. A can also be followed by other accounts, which are called the *followers* of A. Accounts communicate with each other via tweets, each of which is a short unstructured text no longer than 140 characters. Each account creates a new tweet by updating its status. Twitter is designed as a broadcast medium. Thus each tweet is visible to and can be retweeted, i.e. reposted as a status update, by any other account. Account-To-Account communication is also supported by Twitter via Direct Message (DM) and @-tweets. DM allows one account to send messages to another account via a private channel. In other words, a DM is only visible to its sender and receiver accounts. @-tweets are tweets which include a leading "@" symbol to the intended account (e.g. @VTIS). @-tweets allow an account to inform a tweet to specified accounts directly. Unlike DM, @-tweets are public, i.e. remain visible to all other accounts.

We consider setting up an array of accounts for VTIS on Twitter, which are referred as VTIS-accounts. The name for each VTIS-account is textually related to VTIS, e.g. having VTIS as the common prefix. Regular Twitter accounts are able to follow any particular VTIS-account. By following one or multiple VTIS-accounts, regular Twitter accounts essentially make a subscription. For instance, suppose that we have a VTIS-account named VTIS_Bus12_East, any account following VTIS_Bus12_East will receive information related to CTA Bus Route 12, east bound, which is equivalent to the effect possibly achieved by explicitly subscribing to Bus 12 information.

In addition, the subscription by following VTIS-accounts provides the desirable broadcast delivery in nature. That is to say, if a VTIS-account once posts a tweet, then each of its followers gets the message. Symmetric to subscription, each regular account is able to publish to a most relevant VTIS-account by mentioning it in the tweets. As it can be seen, a publish/subscribe mechanism is outlined within the context of Twitter.

Specifically, we plan to create the VTIS-accounts using the following scheme. For private transport information, there is a one VTIS-account for each road link. A publication that reports a private transport event E is automatically assigned to VTIS-account X, where E is located on the road link represented by X. For example, publication "there is severe congestion on Halsted St. from Roosevelt St. to 18th St." will be assigned to VTIS accounts that correspond to Halsted St. from Roosevelt Street to 18th Street. Unlike private traveler information, public traveler information is usually queried or indexed by station rather than location. Thus, each station in any public transport mode is mapped to a VTIS-account, e.g. VTIS_BlueLine_Jackson, VTIS_Bus12_Rooselvet, etc. A publication that reports public transport information regarding a station S is assigned to the VTIS-account that corresponds to S. For example, publication "north exit of Jackson Station of Blue Line is closed" will be assigned to VTIS_BlueLine_Jackson. A publication that reports public transport information regarding a vehicle V is assigned to VTIS-accounts that correspond to stations through which V is going to pass in near future. For example, publication "Blue line train is late at Jackson Station for 5 minutes" will be assigned to all VTIS-accounts that correspond to the stations of Blue Line that are downstream relative to Jackson Station. Figure 2 shows an example scenario in which traveler information is shared among travelers via Twitter.

Each private or public route can be considered as a connected sequence of road links or transfer stations respectively. Since each subscription is a route, we can match a subscription with publications by associating the subscription to relevant VTIS-accounts. In other words, when a subscriber submits a route R, VTIS will automatically make the subscriber follow the VTIS-

accounts that correspond to the road links or/and stations covered by the route R. By this means, the subscriber can easily receive real-time traveler information pertaining to his/her defined route.

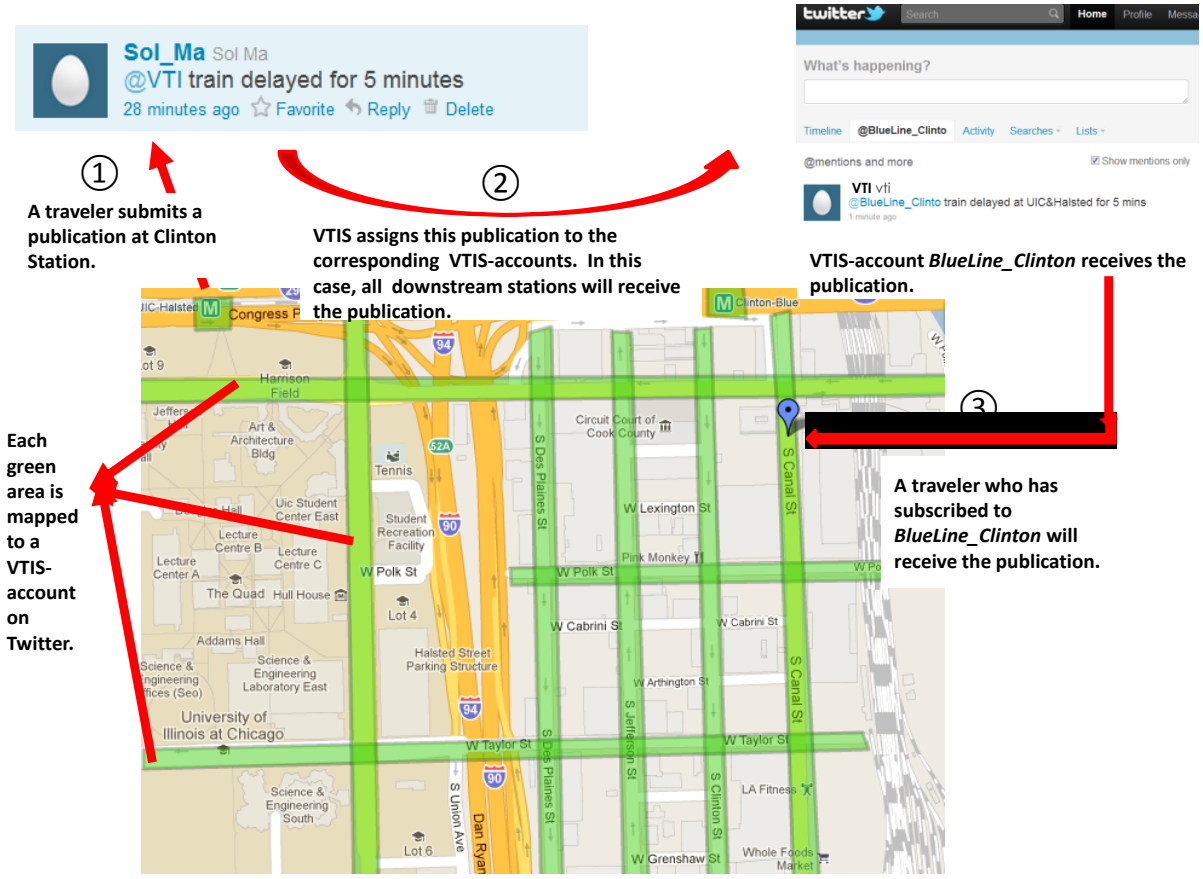


Figure 41 An example scenario for sharing traveler information in VTIS via Twitter

5.4.2 Publication and Subscription Format

In this subsection, we specify publication and subscription formats in the VTIS system. For the purpose of publishing, natural language would be the ideal approach for most travelers due to simplicity. We will exploit current available speech recognition software to convert spoken words into text. Each publication describes certain specific type of traveler information. Each type of traveler information usually has a frequently used bag of keywords which together describe the essential detail constraints placed on the information. All types and their corresponding bag of keywords (separated by comma) are listed in Table 8.

Table 8 Event Types, their associated key words and examples

CATEGORY	TYPE	KEYWORDS ([] indicates optional)	EXAMPLE INSTANCE	EXAMPLE INSTANCE INTERPRETED IN NATURAL LANGUAGE
Public Traveler Information	DELAY	[Mode], Route, Bound, Stop, [+]Delay	Bus, 12, East, Morgan&Roosevelet, 10; Train, Blue Line, O'Hare, UIC&Halsted, +5;	CTA bus Route 12 east bound is 10 mintues late at Morgan&Roosevelet; Blue Line, O'Hare bound is already 5 minutes late at UIC stop and still does not appear
	CLOSING	Stop, [End_Time]	Blue Line UIC-Halsted, Today 3 pm;	Blue Line UIC stop is closed until today 3pm.
	EMERGENCY	[Mode], Route, [Bound], Location, Description	Bus, 12, East, Morgan, a passenger blacked out; Train, Blue Line, O'Hare, Jackson, robbery crime	A passenger blacked out on Bus Route 12 east bound at Morgan street; A robbery crime happens on Blue Line to O'Hare at Jackson
Private Traveler Information	ACCIDENT	Location, Description	I94 Exit 23, a rear-end car accident	a rear-end car accident happens at I94 Exit 23
	CONSTRUCTION	Road_Segment, [End_Time]	Lakeshore Drive (Washington to Chicago), 11/11/11;	Road maintenance on Lakeshore Drive from Washington to Chicago until 11/11/11
	CONGESTION	Road_Segment, Severe/Heavy/Moderate	Lakeshore Drive (Washington to Chicago), Moderate	Traffic on Lakeshore Drive from Washington to Chicago experiences a moderate congestion
	POTHOLE	Road_Segment, Severe/Heavy/Moderate	Taylor between Halsted and Ashland, Severe	Taylor St. between Halsted and Ashland has a severe pothole problem.
	LIGHT	Intersection	Taylor&Morgan	Traffic lights at intersection Taylor&Morgan is down
	EVENT	Event_Name, [Description], Location, [Duration]	Cubs game, Baseball game, Cellular Field, 7-11 pm tonight	There is a Cubs baseball game at Cellular Field tonight 7-11 pm.
	WEATHER	Weather, Area, [Duration]	Blizzard, North Chicago, tonight 8-midnight	There will be a blizzard covering Northern Chicago tonight from 8-12pm
	PARKING	Parking_Lot_Name, 0/1/2/3	UIC Parking Lot 5, 3	UIC Parking Lot 5 is at the fullest level.
	POLICE	Location	Halsted Street	Police cars appear on Halsted Street
	CAMERAS	Intersection	Halsted&Roosevelt	Camera installed at Halsted&Roosevelt

We postulate that the most common subscription from travelers is a route. Travelers can define a route in VTIS by 2 steps. Firstly a traveler inputs the origin and destination (OD) pair of a route via the interface provided by VTIS. The VTIS system then will invoke a navigator or trip planner

service to calculate possible route choices for the given OD pair. The traveler completes defining a route by selecting one of them. Figure 42 shows a gallery of the screenshots of the current mobile clients for VTIS.

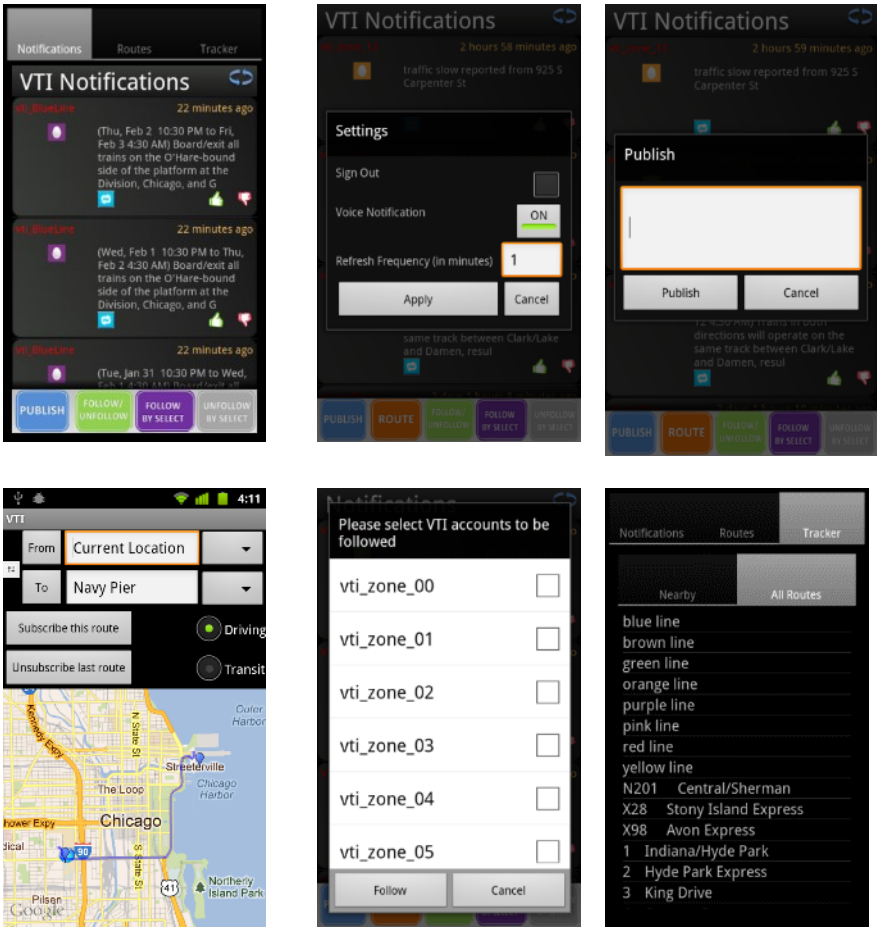


Figure 42 A gallery of the screenshots of the current mobile clients for VTIS

5.4.3 Evaluation of the Prototype

We expect that some students will provide us first-hand user experiences as well as comments and feedback on both the functionally and the user interface of the application. The source code will be released to the students as well. And they are strongly encouraged to improve the prototype implementation by discovering and eliminating bugs.

At second stage, we plan to release our application to the public via App stores, such as Android Market or IPhone App store. Average travelers are then accessible to our application and help us

improve it. We will explore piggybacking VTIS on an existing system developed at UIC, namely TransitGenie (see [22]). TransitGenie provides routing on public transportation, and has thousands of existing users.

Chapter 6

UPDetector: Sensing Parking/Unparking Activities Using Smartphones

6.1 Introduction

Vacant parking spaces are scarce resources in many urban areas. Finding vacant parking spaces in the crowded urban environment can be very frustrating. In addition, cruising for a vacant parking space slows down traffic, causes traffic jams, and pollutes the environment. It is reported that vehicles searching for parking in downtown Los Angeles created 38 trips around the world, producing 730 tons of carbon dioxide and burning 47,000 gallons of gas in one year [102].

Real-time parking space availability information is of great value in alleviating this problem. By feeding such information to navigation systems, drivers can be directly led to an available parking space. Existing approaches of generating/collecting real-time parking spaces availability information can be classified into four categories: (i) infrastructure based; (ii) probe vehicle based; (iii) prediction based; and (iv) participatory sensing based.

An infrastructure based approach requires installing sensors under the pavement (e.g. SFPark [4] project in San Francisco and StreetLine sensors). This is expensive to implement and maintain. For example, the SFPark project costs $23M. Furthermore, the sensors tend to malfunction in adverse weather conditions, e.g. when covered by mud or snow.

A probe vehicle based approach [80] uses vehicles equipped with inexpensive sensors such as ultrasonic sensors to scan the street. To provide real-time availability information, a probe vehicle needs to scan the same street repeatedly; and to cover large areas, multiple vehicles need to scan different streets concurrently. Thus, this approach incurs a high cost and, if dedicated vehicles are used, introduces additional traffic.

A prediction based approach focuses on inferring the availability information by combing historical parking information with parking activities detected in real-time. In [5], authors use Kalman filter to integrate the historical parking availability information with real-time parking/unparking activities detected by smartphones to infer the current parking availability. This approach relies on other approaches to collect real-time parking slot availability information. Some indicators of parking are mentioned in [104], but a comprehensive list and fusion methods are not provided.

Participatory sensing based approach exploits the sensors in smartphones to detect parking/unparking activities. We design and implement an energy-efficient mobile App called Unparking/Parking detector (*UPDetector)* that effectively detects parking and unparking activities by analyzing and fusing the data of multiple sensors embedded in smartphones. Specifically, the contributions of this section include:

- We propose several indicators, each associated with one or more smartphone sensors, for detecting parking/unparking activities. These indicators cover both paid and free parking scenarios. For the purpose of energy conservation, we distinguish between periodical and triggered indicators.
- We propose a probabilistic method to fuse features output by different indicators. These indicators are asynchronous, i.e. they output feature-vectors at different times. The proposed fusion method is proved to have a desirable *reinforcement* property. The fusion method can be applied to inferring other high-level human activities that are characterized by multiple asynchronous indicators. For example, detecting if a driver is fueling at a gas station is such an activity.
- The proposed detection method works regardless how the phone is placed, e.g. in the shirt pocket or in a handbag.

- Our design of *UPDetector* reduces the usage of GPS to save power. We evaluate its energy-consumption via experiments.

The rest of this section is organized as follows. In Sec. 6.2, we introduce indicators and describe how to fuse features output by different indicators. Next we detail the features and implementation of individual indicators in Sec. 6.3. In Sec. 6.4 we conduct experiments to show performances of the implemented *UPDetector* App. Related work is presented in Sec. 6.5, and in sec.VI we conclude and discuss the future work.

6.2 Indicators and Indicator Fusion

In this section, we first propose a list of indicators in subsection 6.2.1. Then, in subsection 6.2.2, we divide the indicators into two types, the periodical and the triggered indicators. Finally, we propose a fusion method in subsection 6.2.3.

6.2.1 Preliminaries on Indicators

An *indicator* is an event that reveals some hint or clue of a parking or an unparking activity. For example, one indicator for unparking is that a person first walks then drives. But this indicator cannot distinguish a passenger from a driver. From this perspective, a stronger indicator for unparking is that the phone is connected with the car via Bluetooth since in general a passenger is less likely than a driver to connect to the car via Bluetooth. Similarly, Bluetooth disconnection from the car is an indicator for parking. Another exemplary indicator for parking is that a person walks towards a roadside pay box and then walks back to a car. Note that this *pay-at-street-parking* indicator is complex enough to be considered an activity by itself and thus can be decomposed into sub-indictors in order to be implemented.

Table 9 gives a list of indicators, where the second column states whether the indicator is for parking, unparking or both activities; and the last column lists the sensors that are required to implement the indicator.

Indicators output *vectors.* Each vector consists of multiple scalar values, each of which is called a *feature*. Vectors of the same indicator have the same set of features. For example, the features of the acoustic sound indicator include Zero Crossing, Spectral Flux [6]. Sec. 6.3 details the features of a subset of the indicators listed here.

Table 9 Example indicators of parking/unparking activities

Indicator	Activity	Explanation	Sensors
change in the variance of the acceleration (*CIV*)	both	In parking activities, a person first drives and then walks. Since walking often has a large variance in acceleration while driving has a small variance, this transition leads to a sudden increase in the variance of acceleration. Likewise, in unparking activities, the variance of acceleration usually suddenly decreases.	accelerometer
phone connected or disconnected to the car via Bluetooth	both	The phone is connected/disconnected to a car via Bluetooth. The App asks the user to identify the car Bluetooth device from a list of available Bluetooth devices. This request is only done once.	Bluetooth
motion-state transition (MST)	both	A parking activity corresponds to a transition from the *driving* state to the *walking* state; and an unparking activity corresponds to a transition from the *walking* state to the *driving* state.	accelerometer
acoustic signals	both	The sounds of human-vehicle interactions that are typically made only during parking or unparking activities. Example interactions include turn on/off the vehicle engine, open and close the vehicle doors.	microphone
car backing	both	Backing the car is common in parking/unparking activities. It is detected by sensing a sudden reverse in the direction of acceleration.	accelerometer and gyro
pay at street-parking box	parking	In the paid street parking scenarios, a driver often needs to walk to a pay box to buy a parking ticket and walk back to the car to place the ticket in the car.	accelerometer, gyro and GPS
parking payment mobile App's	parking	Parking payment mobile App's such as ParkMobile[7] and PayByPhone[8] give a hint of a possible parking activity when such an App is brought to the foreground of the smartphone by a user.	
Wi-Fi signature [9]	unparking	If the parking location is known, a Wi-Fi signature can be created for it and then used to detect an unparking activity by periodically comparing the signature of the current location to that of the known parking location.	wireless interface

6.2.2 Periodical and Triggered Indicators

Indicators that rely only on energy-efficient sensors (e.g. the accelerometer) output a vector periodically. Such indicators are referred to as *periodical* indicators. For example, both the *Change-In-Variance* indicator and the *motion state transition* indicator are periodical indicators.

Indicators that involve energy-hungry sensors such as the microphone, are not periodically monitored for the purpose of conserving energy. They are triggered to output only when the parking or unparking becomes the *hypothesis*, i.e. indicated by the periodical vectors as the most likely outcome among the three, namely *parking*, *unparking*, *none*. We refer to such indicators as *triggered* indicators. For example, the engine- start sound is a triggered indicator. That is, only when the periodical vectors indicate unparking as the most likely outcome, the microphone starts to record a few seconds and output a vector of features of the recorded sound sample. Triggered indicators can be considered auxiliary evidences to verify or refute the hypothesis proposed by the periodical indicators.

Table 10 lists the indicators described in Table 9 with their corresponding category, i.e. periodical or triggered. The table shows the output frequency for the periodical indicators; and for the triggered indicators, it shows which hypotheses, i.e. parking, unparking or both, trigger the indicator.

Table 10 List of categorized indicators

Indicator	Type
sudden change in the variance of acceleration	periodical: once every few seconds
phone connected or disconnected via Bluetooth	periodical: frequency at which the smartphone monitors the Bluetooth connection
motion state transition	periodical: once every few seconds
acoustic signals	triggered: by parking and unparking hypotheses
car backing	periodical (only when the user at *in_vehicle* state): once every a few seconds
pay at street-parking	triggered: by parking hypothesis
parking payment mobile App's	periodical: frequency at which the smartphone monitors the foreground App

Wi-Fi signature[9]	periodical: compute the Wi-Fi signature at certain frequency and compare it to the signature of the parking location

6.2.3 Indicator Fusion

In this section, we first describe the proposed fusion method. Then we prove that our fusion method boosts the confidence in the detected result when compared to a single indicator.

6.2.3.1 Proposed Fusion Method

Whenever some indicator outputs a vector, we need to calculate the probability for each of the three possible outcomes, i.e. *parking*, *unparking*, and *none*, denoted by O_1, O_2, O_3, respectively. Let d be the average duration of a parking/unparking activity (e.g. one minute). Assume that at time point t a vector P of periodical indicator I is generated. We first collect P and the latest vector P' of every periodical indicator other than I, (assuming that vector P' is generated no earlier than time point t-d) into a vector set S. Each indicator is considered independent and thus the vectors in S are independent. Define *fusion set* as the set of independent vectors to be fused. S is an example fusion set. Therefore, we can compute the probability $P(O_i|S)$, i=1,2,3 using Eq. (6.1) below. The calculation of the term $P(X|O_i)$ and $P(O_i)$ are detailed in subsections b) and c) respectively.

$$P(O_i|S) = \prod_{X \in S} P(X|O_i) * P(O_i)/P(S) \quad (6.1)$$

If *none* is the most likely outcome, then no indicator is triggered and thus no parking or unparking activity is detected. Otherwise, the most likely outcome (i.e. either parking or unparking), denoted by O_h, becomes the *hypothesis*, and invokes triggered indicators. Each triggered indicator outputs one vector. Denote such triggered vectors by R_1, R_2,…, R_m, where R_i is generated earlier than R_j for $i < j$. Then the hypothesis O_h is tested in the following way. Let $\mathcal{R}_j$ the set of vectors including the triggered vector R_j and all triggered vectors that are generated before R_j and the periodical vector set S that proposes the hypothesis, i.e. $\mathcal{R}_j = S \cup \{R_i | i \leq j, i \in [1, m]\}$. Whenever vector R_j is generated, we use Eq. (1) to calculate the probability for all three

outcomes, where set S is replaced by set $\mathcal{R}_j$. That is, $\mathcal{R}_j$is a fusion set. Then we normalize the calculated probabilities, denoted by $P_N(O_i|\mathcal{R}_j)$, using Eq. (6.2).

$$P_N(O_i|\mathcal{R}_j) = \frac{\prod_{X\in\mathcal{R}_j} P(X|O_i)*P(O_i)}{\sum_{i=1}^{3}(\prod_{X\in\mathcal{R}_j} P(X|O_i)*P(O_i))}, i = 1,2,3 \quad (6.2)$$

We set a threshold $T \in (0,1)$, referred to as the *detection threshold*, such that an activity of the hypothesis outcome, i.e. parking or unparking, is considered detected only when the normalized probability of the hypothesis is above T, i.e. $P_N(O_h|\mathcal{R}_j) \geq T$. Note that once a parking or an unparking activity is detected, we say that hypothesis O_h is verified by vector set $\mathcal{R}_j$. Therefore, there is no need to consider all triggered vectors that are generated after R_j, i.e. R_k, $k = j+1, j+2, \dots m$.

Multiple detections of the same activity: It is possible that one parking/unparking activity is detected multiple times by some indicator (e.g. the *CIV* indicator) because the activity lasts a period in which the indicator outputs multiple times. If so, we consider all detected activities of the same type (i.e. parking or unparking) within a short period (e.g. half a minute) as a single activity.

a) Localization Process

A parking or unparking activity needs to be associated with the time and location of the activity. To save energy, the App only invokes the localization process when it is necessary. The timing for localization could be either when a hypothesis is proposed (by the periodical vectors) or when a hypothesis is confirmed. If the location is retrieved at the time when a hypothesis is proposed, then the location is cached upon retrieval, and consumed if later the hypothesis is confirmed.

Define the temporal interval from the time when a parking/unparking activity happens to the time when a location is retrieved for the activity as the *delay of localozation*, or simply, the *delay*. Obviously the smaller the delay is, the closer the retrieved location is to the true location where the activity happens. Thus, it is better to invoke the localization process at the time when a hypothesis is proposed instead of confirmed since it leads to a smaller delay.

The App uses the following localization process. A location fix L_1 is retrieved via the smartphone localization API (e.g. in Android, the localization API intelligently chooses the best location source among GPS, WiFi, and cellular networks). Meanwhile another location L_2 is requested via Skyhook [5], a third-party location provider which uses known Wi-Fi hotspots to localize. We then choose the location with the higher accuracy as the detected location of the hypothesis. In our experiments, we observe that the localization process takes only about two seconds on average.

If one parking/unparking activity is detected multiple times we use the time and location of the first detection.

b) Calculation of $P(X|O_i)$

In this subsection, we detail how to calculate $P(X|O_i)$, i.e. the probability that vector X occurs given outcome O_i. Let $X = (x_1, x_2, \dots, x_n)$ where x_i is a feature. We regulate that all features in X are mutually independent. In the case that some features are dependent of each other (two features are dependent if their Pearson's Correlation is over a threshold), only one of them is included in X. Since all the features in X are mutually independent, the term $P(X|O_i)$ can be computed by Eq. (6.3), where $P(x_k|O_i)$ is the probability that feature x_k has the current value given the outcome is O_i.

$$P(X|O_i) = \Pi_{k=1}^{n} P(x_k|O_i) \tag{6.3}$$

The term $P(x_k|O_i)$ is estimated using the following approach. Conduct experiments that generate the O_i outcome. From these experiments collect a sample set of x_k's under the O_i outcome. Normalize all collected x_k's into the $[0,1]$ interval. Discretize the range [0, 1] into several bins, allocate the collected samples into the corresponding bins based on the value of x_k, and calculate the frequency of each bin. If the frequencies of the bins approximate a normal distribution, we estimate the $P(x_k|O_i)$ using the normal distribution of which the mean and standard deviation

are estimated using the collected x_k samples. Otherwise, $P(x_k|O_i)$ is estimated to be the frequency of the bin in which x_k falls; if a new x_k falls outside [0, 1] after normalization, then $P(x_k|O_i) = 0$.

Note that some $P(x_k|O_i)$'s are estimated rather than obtained by experiments. For example, for the Bluetooth indicator, we estimate how many drivers have smartphones connected to the car via Bluetooth, instead of conducting experiments to determine it.

c) Estimating $P(O_i)$'s

Prior probabilities $P(O_i)$'s are estimated using the following approach. $P(O_1)$ of a specific user U is equal to (the amount of time spent on parking activities per *statistical window* / the amount of time per *statistical window*). The size, i.e. amount of time, of a *statistical window* is dependent on the location and time of the day. For example, if user U enters a parking structure (detected by using the energy-efficient Wi-Fi signature method [9]), it means that user U is likely to park soon and thus the statistical window may be just a few minutes. For another example, the size of the statistical window is larger during the night than during daytime since generally parking activities are less likely to occur at night than in daytime. In the case that there is more than one rules that determine the current window size, the smallest window will apply. For example, assuming that the size of the window is eight hours at night and is ten minutes if the car just enters the garage, then the window is ten minutes if a car enters a garage at night. To estimate the amount of time spent on the parking activities during a statistical window for user U, the App needs to count the average number of parking activities and estimate the average time duration that each parking activity takes. At the beginning, when the user just starts to use our App and there are not enough samples to calculate these, default values will be applied.

Similarly, $P(O_2)$ is equal to (the amount of time spent on unparking activities per *statistical window* / the amount of time per *statistical window*). Finally, $P(O_3) = 1 - P(O_1) - P(O_2)$.

6.2.3.2 Reinforcement Property

Our proposed fusion method has a nice *reinforcement* property. That is, via combining vectors from different indicators of which each most likely occurs under the hypothesis (i.e. either a parking/unparking outcome), the fusion provides us a higher confidence in the hypothesis outcome. First let us prove that the reinforcement property holds when applied to two indicators. We formulize the property as follows. The formulation uses the parking outcome as the hypothesis, but the same argument stands for the unparking hypothesis.

Theorem: *Given two vectors* X_1, X_2 *in a fusion set such that* $P(X_i|O_1) > Max\big(P(X_i|O_2), P(X_i|O_3)\big), i = 1,2$ *,i.e. both most likely occur under the parking outcome* O_1 , $P_N(O_1|X_1, X_2) > Max(P_N(O_1|X_1), P_N(O_1|X_2))$.

Proof: Since X_1, X_2 are interchangeable, next we only prove $P_N(O_1|X_1, X_2) > P_N(O_1|X_1)$. While $P_N(O_1|X_1, X_2) > P_N(O_1|X_1)$ can be proved using the same rationale. Using the definition of normalized probability as define by Eq. (6.2), we have Eq. (6.4).

$$\begin{cases} P_N(O_1|X_1, X_2) = \frac{P(O_1|X_1, X_2)}{P(O_1|X_1, X_2)+P(O_2|X_1, X_2)+P(O_3|X_1, X_2)} \\ P_N(O_1|X_1) = \frac{P(O_1|X_1)}{P(O_1|X_1)+P(O_2|X_1)+P(O_3|X_1)} \end{cases} \tag{6.4}$$

Thus, substituting Eq. (6.4) into $P_N(O_1|X_1, X_2) > P_N(O_1|X_1)$, we obtain Eq. (6.5).

$$\frac{P(O_1|X_1, X_2)}{P(O_1|X_1, X_2)+P(O_2|X_1, X_2)+P(O_3|X_1, X_2)} > \frac{P(O_1|X_1)}{P(O_1|X_1)+P(O_2|X_1)+P(O_3|X_1)} \tag{6.5}$$

Taking the reciprocals on the both sides of Eq. (6.5), we get Eq. (6.6)

$$\frac{P(O_1|X_1,X_2)}{P(O_1|X_1, X_2)} + \frac{P(O_2|X_1,X_2)}{P(O_1|X_1, X_2)} + \frac{P(O_3|X_1,X_2)}{P(O_1|X_1, X_2)} < \frac{P(O_1|X_1)}{P(O_1|X_1)} + \frac{P(O_2|X_1)}{P(O_1|X_1)} + \frac{P(O_3|X_1)}{P(O_1|X_1)} \tag{6.6}$$

Subtracting one from both sides of Eq. (6.6), we get Eq. (6.7).

$$\frac{P(O_2|X_1,X_2)}{P(O_1|X_1, X_2)} + \frac{P(O_3|X_1,X_2)}{P(O_1|X_1, X_2)} < \frac{P(O_2|X_1)}{P(O_1|X_1)} + \frac{P(O_3|X_1)}{P(O_1|X_1)} \tag{6.7}$$

A sufficient condition for Eq. (6.7) is Eq. (6.8). That is, if Eq. (6.8) holds then Eq. (6.7) stands, and thus the theorem is proved.

$$\begin{cases} \frac{P(O_2|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(O_2|X_1)}{P(O_1|X_1)} \\ \frac{P(O_3|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(O_3|X_1)}{P(O_1|X_1)} \end{cases} \tag{6.8}$$

Next we prove the first equation in Eq. (6.8), i.e. $\frac{P(O_2|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(O_2|X_1)}{P(O_1|X_1)}$. The other part $\frac{P(O_3|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(O_3|X_1)}{P(O_1|X_1)}$ can be proved using the same rationale. Denote $\frac{P(O_2|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(O_2|X_1)}{P(O_1|X_1)}$ by Eq. (6.9).

$$\frac{P(O_2|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(O_2|X_1)}{P(O_1|X_1)} \tag{6.9}$$

Using the Bayes rule on the right side of Eq. (6.9), we have Eq. (6.10).

$$\frac{P(O_2|X_1,X_2)}{P(O_1|X_1,X_2)} < \frac{P(X_1|O_2)}{P(X_1|O_1)} \times \frac{P(O_2)}{P(O_1)} \tag{6.10}$$

Using the Bayes rule on the left side of Eq. (6.10), we have Eq. (6.11).

$$\frac{P(X_1|O_2,X_2)}{P(X_1|O_1,X_2)} \times \frac{P(O_2|X_2)}{P(O_1|X_2)} < \frac{P(X_1|O_2)}{P(X_1|O_1)} \times \frac{P(O_2)}{P(O_1)} \tag{6.11}$$

Using the Bayes rule on the second term of the left side of Eq. (6.11), we have Eq. (6.12).

$$\frac{P(X_1|O_2,X_2)}{P(X_1|O_1,X_2)} \times \frac{P(X_2|O_2)P(O_2)}{P(X_2|O_1)P(O_1)} < \frac{P(X_1|O_2)}{P(X_1|O_1)} \times \frac{P(O_2)}{P(O_1)} \tag{6.12}$$

Dividing the common factor $\frac{P(O_2)}{P(O_1)}$ from the both sides of Eq. (6.12), we have Eq. (6.13).

$$\frac{P(X_1|O_2,X_2)}{P(X_1|O_1,X_2)} \times \frac{P(X_2|O_2)}{P(X_2|O_1)} < \frac{P(X_1|O_2)}{P(X_1|O_1)} \tag{6.13}$$

Since indicators are independent, X_1 and X_2 are independent, Eq. (6.13) is simplified to Eq. (6.14).

$$\frac{P(X_1|O_2)}{P(X_1|O_1)} \times \frac{P(X_2|O_2)}{P(X_2|O_1)} < \frac{P(X_1|O_2)}{P(X_1|O_1)} \tag{6.14}$$

Since X_2 most likely occurs under O_1, i.e. $\frac{P(X_2|O_2)}{P(X_2|O_1)} < 1$. Then it is clear that Eq. (6.14) stands and thus Eq. (6.8) stands. Therefore, we have $P_N(O_1|X_1,X_2) > Max(P_N(O_1|X_1), P_N(O_1|X_2))$ ▪

Similarly, it is easy to prove the general case, i.e. that when $i = 1\ or\ 2,\ P_N(O_i|X_1,X_2,\dots X_m) > Max(P_N(O_1|X_1),\dots P_N(O_1|X_m))$ given X_j's are independent to each other and $P_N(X_j|O_i), j = 1,2,\dots m$ is the largest among $P_N(X_j|O_k)$, k=1,2,3.

6.3Implementation of Individual Indicators

In this section, we detail the features and implementation of only a subset of the proposed indicators. Other indicators, once implemented, can be plugged-and-played into our system through the fusion method described earlier.

6.3.1 Change-In-Variance (CIV) Indicator

6.3.1.1 Preliminaries on Accelerometer

The accelerometer of an android phone has a coordinate-system consisting of three axes, as shown in Figure 43. The *X* axis is horizontal and points to the right, the *Y* axis is vertical and points up and the *Z* axis points towards the outside of the front face of the screen. Each reading of the accelerometer contains three values, that is, one value for each axis. The three axes are defined relative to the screen of the phone in its default orientation. The *X* axis is horizontal and points to the right, the *Y* axis is vertical and points up and the *Z* axis points towards the outside of the front face of the screen. The resultant acceleration (or simply the acceleration) refers to a single value *Accel* that normalizes the accelerometer readings along the three axes using Eq. (6.15).

$$Accel = \sqrt[2]{A_X^2 + A_Y^2 + A_Z^2} \tag{6.15}$$

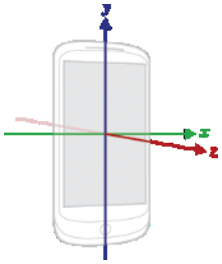


Figure 43 Axes of mobile phone

6.3.1.2 Features of the CIV Indicator

A sliding window with a fix size equals to W seconds is used. The sliding window moves forward N seconds every time it slides. That is, two consecutive windows overlap W-N seconds. We refer to N as the sliding step thereafter. During each window, we calculate the difference between the variance of the acceleration within the second half and the variance of the first half of

the window. Hereafter we refer to this feature as the *VariDiff* of a window. Intuitively, as shown in Figure 44, an unparking activity results in a window where the first half (corresponding to the *walking* state) has a large variance while the second half (corresponding to the *driving* state) has a small variance. Likewise, the parking activity has a similar sharp contrast in the variance between the two halves of the window. We have observed via experiments that this variance discrepancy between two halves of the window exists no matter where the phone is placed, e.g. in pant leg pocket, in handbag, etc.

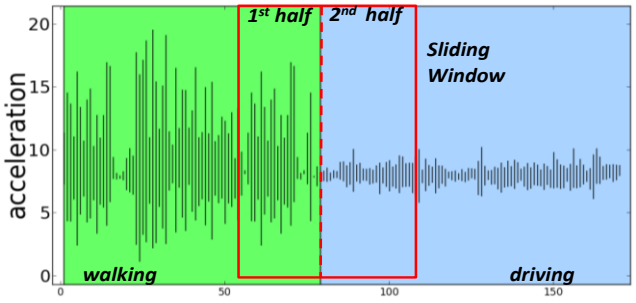


Figure 44 The sliding window for the CIV indicator

However, the *VariDiff* feature of the current window itself is not sufficient due to noise. One example of noise is that the *VariDiff* feature of a window during which the user walks, may be either positive or negative since the acceleration during walking is oscillating. As a result, such a window may be misidentified as a parking or unparking activity. To deal with the noise, we consider using the *VariDiff* feature of all windows within a *scope*. Denote the scope by S. Then each *CIV* vector consists of three features: (i) the *VariDiff* feature of the current window; (ii) the average value of the *VariDiff* feature during the preceding $S/2$ windows; (iii) the average value of the *VariDiff* feature during the succeeding $S/2$ windows. Formally, the *scope* S is the total number of preceding and succeeding windows that are considered in a *CIV* vector. Observe that S does not include the current window. In order to calculate feature (iii), production of the vector of a window is delayed for $S/2$ windows. Intuitively, a vector that corresponds to a parking activity has feature

(i) being positive and feature (ii) being close to zero. Similarly, a vector that corresponds to a unparking activity has feature (i) being negative and feature (iii) being close to zero.

Figure 45 illustrates the calculation of the *CIV* vectors. Assume that the *VariDiff* feature of 1st, 2nd, 3rd, 4th, 5th window has a value of 0.03, 0.01, 2.6, 1.6, 1.8, respectively, and the scope *S* equals to 4. Then when the 5th window ends, the *CIV* vector of the 3rd window is computed and equals to (0.02, 2.6, 1.7), where 0.02 is the mean of the *VariDiff* of the 1st and 2nd windows and 1.7 is the mean of the *VariDiff* of the 4th and 5th windows.

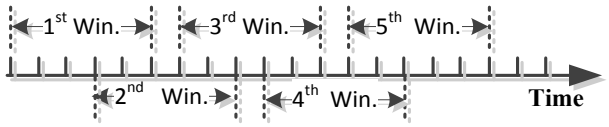


Figure 45 An example of calculating CIV vectors

6.3.2 Bluetooth Indicator

The Bluetooth embedded in the phone is a strong indicator when it is enabled. When a phone is connected to a car (requiring one time human input to indicate the name of the Bluetooth device of the car), there is a good chance that the person is the driver and intends to depart, which corresponds to an unparking activity. Similarly when a phone is disconnected, it is likely due to a person leaving the car, which indicates a parking activity.

There is only one feature for the Bluetooth indicator, which takes one of the following three values: i.e. *connected, disconnected, not enabled*.

6.3.3 Motion State Transition Indicator

Motion states, such as *waling, driving*, etc., can be classified from raw accelerometer readings. After motion states are classified, the transitions between motion states are identified to signify the parking/unparking activities. Figure 46 (a) and (b) shows the motion state transition for parking and unparking, respectively.

(a) parking (b) unparking

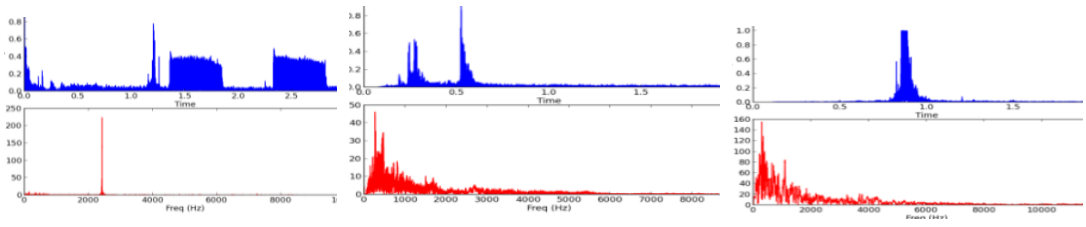
Figure 46 Transitions for parking and unparking, respectively

We implement the classification algorithm described in [12] to classify motion states. Specifically, the classifier outputs a probability distribution over all possible motion states (namely *driving, walking, still*, *sitting* and *standing*) every five seconds using the accelerometer data in the past five seconds.

The vectors of the motion state transition indicator thus include four features. The first two features are the probability of the latest motion state being *walking* and *driving*, respectively. Similarly the last two features are the probability of the second latest motion state being *walking* and *driving,* respectively.

6.3.4 Acoustic Indicators

Acoustic indicators refer to the sounds of human-vehicle interactions that are typically made only during parking or unparking activities. Example interactions include turn on/off the vehicle engine, open and close the vehicle doors. Such sounds often have distinct frequency and amplitude from each other, as shown in Figure 47.



(a) Engine start (b) Open door (c) Close door

Figure 47 Normalized amplitude and the corresponding FFT result of different sound samples

It has been shown that the sounds of these particular human-vehicle interactions can be classified with relative high precision and recall [91]. However, in [91] the authors do not consider the power concern and thus the sounds are recorded constantly. In our case, in order to save energy, acoustic sounds are modeled as triggered indicators as described in Sec. 6.2.2. That is, they are only

activated and output vectors after the parking or unparking outcome becomes the hypothesis. In addition to the work of [91], we included the "bus noise" (i.e. the bus engine sound with the background noise) as one of the acoustic sounds, and found out that the "bus noise" sound is highly distinguishable from other sounds such as engine start, or door open/close. This will help distinguish a private car trip from a bus trip

6.4Evaluation

We implement a prototype system on the Android platform. This section details the experimental methodology and the results. Specifically, we introduce our experimental setting in subsection 6.4.1. Then we show the performance of the App in subsection 6.5.2.

6.4.1 Experimental Methodology

6.4.1.1 Mobile App Implementation

UPDetector is independent of mobile platforms and thus can be implemented on all mobile platforms, such as Android, Apple iOS and Windows mobile systems. We implement a prototype on the Android platform using the Samsung Galaxy S3, which has a 1 GB RAM and quad-core 1.4 GHz Cortex-A9 processor. Figure 48 shows some screenshots of the implemented prototype.

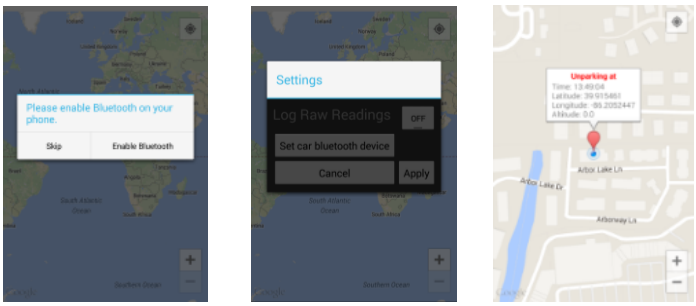


Figure 48 UPDetector implementation screenshots

6.4.1.2 Data Collection

We have implemented the following indicators in the *UPDetector* App: the *Bluetooth* indicator, the *CIV* indicator and the *MST* indicator. We have also implemented the acoustic indicator (i.e. the audio analysis part) on the laptop; unfortunately, we cannot port the function to the App since

Android system currently does not support the audio feature extraction library. The Bluetooth indicator is highly reliable by itself and may work independently of other indicators. Therefore, here we restrict attention to vehicles that do not have Bluetooth devices and present the detection results using the *CIV* and the *MST* indicators.

The time (in seconds) of each parking/unparking activity is manually recorded as the ground truth. The time of an unparking activity is the second when the vehicle starts to move from a parked state; and the time of a parking activity is the second when the vehicle reaches the still state.

The collected data is split into one training set and one test set. The training data set contains 40 parking activities and 40 unparking activities. The test data set contains 60 parking activities and 60 unparking activities. The training set is used to learn the conditional probability of features under different outcomes, i.e. $P(x_k|O_i)$'s. Via experiments, we observe that $P(x_k|O_i)$'s in the training set approximate normal distributions. So we estimate the parameters of the normal distributions using the training data set and then use them for the test data set. The test data is used to evaluate the performance of the detection of parking/unparking activities.

6.4.1.3 Detection Methods

We evaluate five detection methods. These methods are categorized into two groups. The first group consists of three methods that use a single indicator: (i) the method that only uses the *Change-in-Variance* (*CIV*) indicator, referred to as the *CIV* method hereafter; (ii) the method that only uses the *Motion State Transition* (*MST*), where the motion state classifier is implemented using the features described in [63] (multiple classification methods, including meta-classification methods such as AdaBoost and RandomForest, are tried to train the model and the best classification method, i.e. RandomForest, is chosen), referred to as the *MST-CL1* method hereafter; (iii) the method that only uses the *Motion State Transition* (*MST*), where the motion state classifier is provided by Google Activity Recognition *(GAR)* API, referred to as the *MST-CL2* method hereafter. The *GAR* API returns a distribution over five possible states including *driving*, *walking, still, tilting,* and *unknown*. We have observed that the *GAR* API outputs the *unknown* state as the most likely state

frequently. To better utilize the API's results, we modify the most likely state for the following case. If the *GAR* API outputs *unknown* as the most likely state, and if the second most likely state S has a likelihood that is larger than the sum of the likelihoods of other three states except *unknown* and *S*, we treat S as the most likely state.

The second group consists of two methods that fuse multiple indicators. The first method, referred to as the *CIV-MST_CL1* method, combines the *CIV* indicator and *MST-CL1* using the probabilistic based fusion algorithm described in Sec. 6.2.3.

The second method, referred to as the *CIV-MST_CL1_CL2* method, combines the *CIV-MST_CL1* method with the *MST_CL2* method. Specifically, each activity A_d detected by the *CIV-MST_CL1* method is considered a hypothesis (thus a location is retrieved and cached when A_d is detected). We have learned from experiments that the *MST_CL2* method is reliable but suffers from a large delay (see Sec. 6.4.2.1). Therefore, we consider that A_d is confirmed if later *MST_CL2* also detects the same type (i.e. parking/unparking) of activity as A_d. If multiple A_d's of the same activity type have been output by *CIV-MST_CL1* when *MST_CL2* outputs a detection, we use the location retrieved for the first A_d. Note that here we use a simple "and" logic to combine the detection results of *CIV-MST_CL1 and MST_CL2* because *MST_CL2* is highly reliable (but unfortunately has a long delay) and thus a simple "and" logic is sufficient. Otherwise, we would have used the proposed probabilistic fusion algorithm to combine *CIV-MST_CL1* and *MST_CL2*.

6.4.1.4 Matching Detected Activities with the Ground Truth

A detected parking (unparking) activity A_d is matched to a ground truth parking (unparking) activity A_g if the time difference between A_d and A_g is smaller than five seconds[1]. For a detected activity A_d, the matching ground truth activity A_g can always be uniquely identified because any two consecutive ground truth activities are at least minutes away from each other, and thus there is no confusion in the matching.

[1] except for the *MST_CL2* method; since it suffers a long delay, its value is one minute

Note that a ground truth activity may be detected multiple times (i.e. matched to several detected activities that are temporally consecutive and close to each other). This only happens when a sliding window is used in the indicator (e.g. in the *CIV* indicator) and the window slides in a way such that the two consecutives windows overlap. For example, consider the example shown in Figure 44. The red window in the figure represents a detected unparking activity. If we slide the red window slightly to the right, apparently, the new window may still represent a detected unparking activity that is matched to the same ground truth activity. When multiple detected activities are matched to the same ground truth activity A_g, we use the first matched detected activity and ignore the rest of the detected activities that are matched to A_g.

6.4.1.5Performance Measures

The performance is measured by *precision* and *recall*. Eq. (6.16) gives the definition, where *tp*, *fp*, *fn* is the number of true positives, false positives and false negatives, respectively. A detected parking (unparking) activity A_d is a true positive if it matches to a ground truth parking (unparking) activity A_g. That is, the time difference between A_d and A_g is small than five seconds. A_g can always be uniquely identified because any two consecutive ground truth activities are usually at least minutes away from each other and thus there is no confusion in the matching.

$$\begin{cases} precision &= tp \,/\, (tp + fp) \\ recall &= tp/\,(tp + fn) \end{cases} \tag{6.16}$$

Another measure is the delay of localization. Denote by t_g the timestamp of the ground truth activity A_g and by by t_l the time when the location is received. The *delay*, denoted by D, can be calculated using Eq. (17).

$$\boldsymbol{D = t_l - t_g} \tag{6.17}$$

6.4.2 Evaluation Results

6.4.2.1 Detection Accuracy and the Delay

The *MST_CL1* classifier is implemented according to [63]. The *MST_CL2* uses the activity recognition API provided by Google. The API provides one parameter that adjusts the update

frequency. This parameter is set to zero so that the updates are obtained at the highest possible frequency.

Table 11 lists the values for parameters for the *CIV* method and the detection threshold. In this paragraph we discuss these parameters. Many previous works (e.g. [63, 93]) suggest that the window size should be large enough to include a few hundred samples but not too large to increase the delay. Based on the accelerometer sampling frequency in Android (i.e. about 10~30 Hz), 10 seconds is a reasonable window size (other window sizes are also tried and 10 seconds show the best results).

A small sliding step helps capture the sudden change in the acceleration. Intuitively, a small scope helps decrease the delay. We conducted experiments to learn the impact of the *scope* parameter on the precision and recall. The experiments suggest that as the scope *S* increases, the precision and recall first increases then decreases. This is because the scope only helps when it includes recent past samples; and it starts to hurt the performance as it continues to increase and includes samples from a more remote past. The results suggest that a scope of six windows achieves the best precision and recall.

Table 11 Default values of parameters

Notation	Meaning	Value
W	window size of the *CIV* indicator	10 seconds
N	sliding step of the *CIV* indicator	3 seconds
S	scope of the *CIV* indicator	6 windows
T	detection threshold	0.9

Table 12 shows the performance of the detection methods described in 6.4.1.3. Note that the average delay in the table refers to the average delay of the true positives, i.e. the detected activities that are matched to the ground truth activities.

In the first group (i.e. methods that use only one indicator), *MST_CL2* gives the best precision and recall but it suffers from a large delay, especially for unparking activities. For this reason the

MST_CL2 method cannot be used alone (i.e. if used a alone the location of the parking/unparking activity cannot be accurately identified).

Note that the delay for unparking activities is much larger than that for parking activities for *MST_CL2*. This is due to the fact that the *GAR* API outputs *driving* state with a much larger delay than the *walking* state. In comparison, the *CIV* method have a much smaller delay but with a slightly lower recall and a much lower precision. The *MST_CL1* method has the poorest precision and recall among the three methods. Note that the *MST_CL1* uses the features described in [63], where the authors report a much higher precision and recall for human activities classification. But in [63], the phone has a fixed position (i.e. the front leg pocket) while here the phone is placed in various positions. In addition, [63] does not include *driving* as an activity. In general, *driving* is much harder to be correctly classified than on foot activities such as *walking* or *jogging* since *driving* is easily confused with *still* or *standing*. (This may also explain why the *GAR* API outputs *driving* activity with a much larger delay than *walking* activity.) The results of the first group demonstrate that no individual indicator is good enough.

Table 12 Detection accuracy

Detection Methods		Parking Activities			Unparking Activities		
		Recall	Precision	Avg. Delay (secs)	Recall	Precision	Avg. Delay (secs)
Methods that use only one indicator	*CIV*	86.2%	29.7%	10.68	87.9%	45.1%	14.43
	MST_CL1	60.3%	18.6%	20	70.6%	22.2%	14.17
	MST_CL2	94.8%	88.7%	17.75	89.6%	89.6%	46.18
Methods that fuse multiple indicators	*CIV-MST_CL1*	91.3%	23.8%	10.3	96.5%	24.3%	15.72
	CIV-MST_CL1_CL2	93.1%	90.4%	9.98	81.8%	93.1%	14.36

In the fusion method group, the *CIV-MST_CL1* has a higher recall than that of both the *CIV* and *MST_CL1* method. However, the method's precision remains unsatisfying. This is because the fusion process enhances the detection confidence when both the *CIV* and the *MST_CL1* method

correctly detects the same type activity (i.e. parking/unparking) with a low confidence and thus helps improve the recall. However, when both the *CIV* and the *MST_CL1* method mistakenly detect the same type activity with a low confidence, the fusion also boosts the confidence, and as a result the precision of the *CIV-MST_CL1* method may be lower than the largest precision of the constituting methods.

As the integration of the *CIV-MST_CL1* method and the *MST_CL2* method, the *CIV-MST_CL1_CL2* method inherits all the merits: it has a higher precision than both its constituting methods; it has a fairly high recall while keeps a small delay.

6.4.2.2 Energy consumption

We employ PowerTutor [125] to measure the power consumption. For the purpose of localization, GPS is enabled when *UPDetector* is running. But it is in the stand-by mode and consumes little energy (about 0.8 mw) during most of time. GPS only enters the energy-hungry searching mode (about 220 mw) once for each parking/unparking activity. Since there are at most a few parking/unparking activities during a day, the power consumption for localization is negligible.

Most power consumption of the App attributes to CPU usage caused by the computation during the fusion of periodical vectors. When *UPDetector (running the CIV-MST_CL1_CL2 method)* is the only App running and phone activities (such as call, sms) are avoided, the corresponding battery life is about 20.3 hours. The battery life with no app running and no phone activities is around 25 hours. That is, *UPDetector* costs 4.7 hours of the battery life. It is possible to further reduce this cost by decreasing the output frequency of the periodical indicators such as the *CIV,* by special hardware [55], by using Android Geofencing API[2] and by monitoring only outdoors.

[2] Android Geofencing. https://developer.android.com/training/location/geofencing.html

6.5 Related Work

6.5.1 Parking Spaces Detection

In the past, on street parking slot detection is usually performed by sensors embedded in the pavement [4] or in vehicles [80]. However, these efforts require significant investment and are expensive to implement to cover a large city.

Given the proliferation of the mobile devices, recently smartphone applications such as ParkMobile and PaybyPhone[3] that allow drivers to pay for parking by mobile phones are emerging. Such App's can be used by our method as an indicator for parking. [84] proposes a novel method which leverages Wi-Fi beacons in urban environment to detect unparking. This method can be integrated into our work as an indicator for unparking activities. This method by itself is not always applicable since Wi-Fi signature works only when the parking location is covered by multiple Wi-Fi signals.

6.5.2 Activity Recognition

There have been works on detecting motion activities based on readings from sensors in smartphones. Generally, a motion activity detection algorithm is a classifier which reads raw sensor data (e.g. from GPS [31, 105, 129, 130], from accelerometer [36, 82, 113], from both GPS and accelerometer [96] and from Wi-Fi/GSM [83]), processes it to extract features, and then classifies and outputs the motion activity such as *still, walking, running, driving*, etc. The motion state transition based method by itself is not a reliable indicator for parking/unparking detection. We incorporate the *MST* method into our framework which fuses outputs from multiple indicators. Our proposed fusion method is applicable to detect a variety of high level human activities that are more complex than simple motion activities.

[3] Pay By phone http://www.paybyphone.com/how-it-works/

6.5.3 Classifier Fusion

In [110], the authors survey existing methods of combing multiple classifiers. These methods include i) ensemble methods that combine multiple homogenous classifiers (i.e. classifiers that are learned using the same set of features and the same classification algorithm), such as Bagging and Boosting; and ii) non-ensemble methods that combine heterogeneous classifiers, such as the majority voting (e.g. voting based on either the number of each class or the aggregated confidence in each class). We apply the ensemble methods, i.e. the Boosting method via Weka, to implement individual indicators such as *MST*. But the ensemble methods do not handle the asynchronous data problem. That is, in our application scenario, different indicators output vectors of different feature sets at different frequencies. Additionally, we aim to save energy, a consideration that is missing in prior work on classifier fusion. Cost-sensitive boosting [79] methods may be applicable, but it is not clear how to incorporate energy consumption into cost functions. As pointed out by the authors of [110], none of the non-ensemble methods are shown to be superior to others, neither theoretically nor empirically. Our proposed fusion method can be considered a non-ensemble method that is motivated by and designed for the unparking/parking detection application, and potentially applied to other applications with the asynchronous data problem.

In [110], the authors survey existing methods of combing multiple classifiers. However these methods do not apply to our application for two reasons. First those methods assume that the to-be-combined values are categorical while in our case, the output of each indicator has a vector of continuous values. Second, those methods do not handle the asynchronous problem by assuming all classifiers output synchronously. Here the periodical indicators output vectors at different frequencies. In addition, the triggered indicators further complicate the asynchronous problem. And remember that the triggered indicators are introduced to save energy, a consideration that is missing in prior work on classifier fusion.

6.6 Discussion

We presented the design and implementation of a parking/unparking activities detection system called *UPDetector*. We described several indicators and their corresponding features; we proposed a probabilistic fusion method which combines features output by multiple indicators to derive parking/unparking activity detection results. We evaluated the *UPDetector* prototype via experiments, and demonstrated its effectiveness and energy consumption.

Using the implemented *Bluetooth* indicator, the current App we implemented has a certain capability of distinguishing a driver from a passenger. This capability can be further enhanced via incorporating other indicators, e.g. the pay-at-street-parking box indicator and the parking-payment-mobile-App indicator listed in Table 9. In addition, acoustic indicators can be used to distinguish buses from private cars.

In the future, we are going to improve the prediction algorithm presented in [119]. The algorithm is capable of integrating real-time detection information, i.e. activities detected from different devices, with the historical pattern. It outputs parking space availability information that is more reliable than either real-time or historical information alone.

CITED LITERATURE

[1] Bay area rapid transit planner. http://www.bart.gov/.

[2] Intellidrive ╟ safer, smarter, greener. http://www.intellidriveusa.org/.

[3] Regional transit authority trip planner. http://tripsweb.rtachicago.com.

[4] San francisco parking. http://sfpark.org/.

[5] Skyhook Inc. http://www.skyhookwireless.com/.

[6] Slugging. http://en.wikipedia.org/wiki/slugging.

[7] Washington metropolitan area transit authority trip planner.http://www.wmata.com/tripplanner_d/tripplanner.cfm.

[8] Waze, a free, community-based traffic & navigation app. http://www.waze.com/.

[9] Salutation architecture specification 2.0. Tech. rep., Salutation Consortium, June 1999.

[10] Uddi version 2.04 api specification. Tech. rep., OASIS standard, July 2002.

[11] Jini technology core platform specification, v. 2.0. Tech. rep., Sun Microsystems, June 2003.

[12] Upnp device architecture 1.0. Tech. rep., UpnP Forum, Dec. 2003.

[13] Map of slugging sites in washington d.c. *slug-lines.com, Forel Publishing Company, LLC* (June 2010).

[14] AGATZ, N., E.-A.-S. M. W.-X. Sustainable passenger transportation: Dynamic ride-sharing. Tech. rep., Erasmus Research Inst. of Management (ERIM), Erasmus Uni., Rotterdam, 2010.

[15] ANGELACCIO, M., CATARCI, T., AND SANTUCCI, G. Qbd*: a graphical query language with recursion. *Software Engineering, IEEE Transactions on 16*, 10 (oct 1990), 1150 –1163.

[16] ATTANASIO, A., CORDEAU, J.-F., GHIANI, G., AND LAPORTE, G. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Comput. 30*, 3 (Mar. 2004), 377–387.

[17] AYALA, D., WOLFSON, O., XU, B., DASGUPTA, B., AND LIN, J. Parking slot assignment games. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2011), GIS '11, ACM, pp. 299–308.

[18] BADGER, E. Slugging the people transit. *Miller-McCune* (2011).

[19] BALDACCI, R., MANIEZZO, V., AND MINGOZZI, A. An exact method for the car pooling problem based on lagrangean column generation. vol. 52, INFORMS, pp. 422–439.

[20] BAUGH, J. W., KAKIVAYA, G. K. R., AND STONE, J. R. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization 30*, 2 (1998), 91–123.

[21] BERGVINSDOTTIR, K., LARSEN, J., AND JORGENSEN, R. *Solving the Dial-a-Ride Problem using Genetic algorithms*. Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2004.

[22] BIAGIONI, J., AGRESTA, A., GERLICH, T., AND ERIKSSON, J. Transitgenie: a context-aware, real-time transit navigator. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2009), SenSys '09, ACM, pp. 329–330.

[23] BIELLI, M., BOULMAKOUL, A., AND MOUNCIF, H. Object modeling and path computation for multimodal travel systems. *European Journal of Operational Research 175*, 3 (Dec. 2006), 1705–1730.

[24] BUTTYÁN, L., AND HUBAUX, J.-P. Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. Tech. rep., 2001.

[25] CALVO, R. W., DE LUIGI, F., HAASTRUP, P., AND MANIEZZO, V. A distributed geographic information system for the daily carpooling problem. vol. 31, Elsevier Science Ltd., pp. 2263–2278.

[26] CALVO, R. W., AND TOUATI-MOUNGLA, N. A matheuristic for the dial-a-ride problem. In *Proceedings of the 5th international conference on Network optimization* (Berlin, Heidelberg, 2011), INOC'11, Springer-Verlag, pp. 450–463.

[27] CALVO ROBERTO, C. A. An effective and fast heuristic for the dial-a-ride problem. *4OR: A Quarterly Journal of Operations Research 5* (2007), 61–73. 10.1007/s10288-006-0018-0.

[28] CHAN, N. D., AND SHAHEEN, S. A. Ridesharing in north america: Past, present, and future. *Transport Reviews 32*, 1 (2012), 93–112.

[29] CHEN, C., ZHANG, J., COHEN, R., AND HO, P.-H. A trust modeling framework for message propagation and evaluation in vanets. In *Information Technology Convergence and Services (ITCS), 2010 2nd International Conference on* (2010), pp. 1 –8.

[30] CHO, J., SWAMI, A., AND CHEN, I. A survey on trust management for mobile ad hoc networks. vol. PP, pp. 1 –22.

[31] CHU, D., LANE, N. D., LAI, T. T.-T., PANG, C., MENG, X., GUO, Q., LI, F., AND ZHAO, F. Balancing energy, latency and accuracy for mobile sensor data classification. *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11* (2011), 54.

[32] COHEN, E., AND SHENKER, S. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2002), SIGCOMM '02, ACM, pp. 177–190.

[33] CORDEAU, J.-F., AND LAPORTE, G. The dial-a-ride problem: models and algorithms. *Annals of Operations Research 153* (2007), 29–46.

[34] DATTA, A., QUARTERONI, S., AND ABERER, K. Autonomous gossiping: A self-organizing epidemic algorithm for selective. In *In International Conference on Semantics of a Networked* (2004), World, pp. 126–143.

[35] DEMERS, A., GEHRKE, J., HONG, M., RIEDEWALD, M., AND WHITE, W. Towards expressive publish/subscribe systems. In *In Proc. EDBT* (2006), pp. 627–644.

[36] DERNBACH, S., DAS, B., KRISHNAN, N. C., THOMAS, B. L., AND COOK, D. J. Simple and Complex Activity Recognition through Smart Phones. *2012 Eighth International Conference on Intelligent Environments* (June 2012), 214–221.

[37] DESROCHERS, M., LENSTRA, J. K., SAVELSBERGH, M., AND SOURRIS, F. Vehicle routing with time windows: Optimization and approximation. In *VEHICLE ROUTING: METHOD AND STUDIES. STUDIES IN MANAGEMENT SCIENCE AND SYSTEMS* (1988), Elsevier Science, pp. 65–84.

[38] DOTZER, F., FISCHER, L., AND MAGIERA, P. Vars: a vehicle ad-hoc network reputation system. In *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a* (2005), pp. 454 – 456.

[39] DOUCEUR, J. R. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (London, UK, 2002), IPTPS '01, Springer-Verlag, pp. 251–260.

[40] ERIKSSON, J., BALAKRISHNAN, H., AND MADDEN, S. Cabernet: vehicular content delivery using wifi. In *Proceedings of the 14th ACM international conference on Mobile computing and networking* (New York, NY, USA, 2008), MobiCom '08, ACM, pp. 199–210.

[41] EUGSTER, P. T., FELBER, P. A., GUERRAOUI, R., AND KERMARREC, A.-M. The many faces of publish/subscribe. vol. 35, ACM, pp. 114–131.

[42] EUZENAT, J., LOUP, D., TOUZANI, M., AND VALTCHEV, P. Ontology alignment with ola. In *In Proceedings of the 3rd EON Workshop, 3rd International Semantic Web Conference* (2004), CEUR-WS, pp. 59–68.

[43] FIEGE, L., MUHL, G., AND GARTNER, F. C. A modular approach to build structured event-based systems. In *Proceedings of the 2002 ACM symposium on Applied computing* (New York, NY, USA, 2002), SAC '02, ACM, pp. 385–392.

[44] FORTE, A., AND BRUCKMAN, A. Why do people write for wikipedia? incentives to contribute to open-content publishing. In *Group 2005 workshop: Sustaining community: The role and design of incentive mechanisms in online systems. Sanibel Island, FL* (November 2005).

[45] GALIL, Z. Efficient algorithms for finding maximal matching in graphs. In *CAAP'83*, G. Ausiello and M. Protasi, Eds., vol. 159 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1983, pp. 90–113.

[46] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., New York, NY, USA, 1979.

[47] GE, Y., XIONG, H., TUZHILIN, A., XIAO, K., GRUTESER, M., AND PAZZANI, M. An energy-efficient mobile recommender system. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2010), KDD '10, ACM, pp. 899–908.

[48] GERLACH, M. Trust for vehicular applications. In *Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 295–304.

[49] GHOSE, A., IPEIROTIS, P. G., AND SUNDARARAJAN, A. Opinion mining using econometrics: A case study on reputation systems. In *In Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL* (2007).

[50] GIDÓFALVI, G. Instant Social Ride-Sharing. *ITS World, 8p, Transportation Society of America* (2008), 1–8.

[51] GIDOFALVI, G., PEDERSEN, T. B., RISCH, T., AND ZEITLER, E. Highly scalable trip grouping for large-scale collective transportation systems. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology* (New York, NY, USA, 2008), EDBT '08, ACM, pp. 678–689.

[52] GIUNCHIGLIA, F., YATSKEVICH, M., AND SHVAIKO, P. Journal on data semantics ix. Springer-Verlag, Berlin, Heidelberg, 2007, ch. Semantic matching: algorithms and implementation, pp. 1–38.

[53] GOLLE, P., GREENE, D., AND STADDON, J. Detecting and correcting malicious data in vanets. In *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks* (New York, NY, USA, 2004), VANET '04, ACM, pp. 29–37.

[54] GRAHAM, K. Complexity science and social media: Network modeling in following #x201c;tweets #x201d;. In *Systems and Information Engineering Design Symposium (SIEDS), 2010 IEEE* (april 2010), pp. 141 –146.

[55] HAICHEN SHEN, ARUNA BALASUBRAMANIAN, ERIC YUAN, ANTHONY LAMARCA, D. W. Improving Power Efficiency Using Sensor Hubs Without Re-Coding Mobile Apps. Tech. rep.

[56] HUANG, Y., AND GARCIA-MOLINA, H. Publish/subscribe tree construction in wireless ad-hoc networks. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management* (London, UK, 2003), Springer-Verlag, pp. 122–140.

[57] HUANG, Y., AND GARCIA-MOLINA, H. Publish/subscribe in a mobile environment. vol. 10, Kluwer Academic Publishers, pp. 643–652.

[58] HUMPHREYS, L. Mobile social networks and social practice: A case study of dodgeball. *J. Computer-Mediated Communication 13*, 1 (2007), 341–360.

[59] J.-F., C., AND G., L. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological 37*, 6 (2003), 579–594.

[60] JOSANG, A., AND ISMAIL, R. The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference* (2002), vol. 160, Citeseer, p. 324?37.

[61] JOSANG, A., KESER, C., AND DIMITRAKOS, T. Can we manage trust? In *Proceedings of the Third International Conference on Trust Management (iTrust), Versailes* (2005), Springer-Verlag, pp. 93–107.

[62] KAMAR, E., AND HORVITZ, E. Collaboration and shared plans in the open world: studies of ridesharing. In *Proceedings of the 21st international jont conference on Artifical intelligence* (San Francisco, CA, USA, 2009), IJCAI'09, Morgan Kaufmann Publishers Inc., pp. 187–194.

[63] KWAPISZ, J., WEISS, G., AND MOORE, S. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter 12*, 2 (2011), 74–82.

[64] LAI, C., CHANG, H., AND LU, C. C. A secure anonymous key mechanism for privacy protection in vanet. In *Intelligent Transport Systems Telecommunications,(ITST),2009 9th International Conference on* (oct. 2009), pp. 635 –640.

[65] LALOS, P., KORRES, A., DATSIKAS, C. K., TOMBRAS, G. S., AND PEPPAS, K. A framework for dynamic car and taxi pools with the use of positioning systems. In *Proceedings of the 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns* (Washington, DC, USA, 2009), COMPUTATIONWORLD '09, IEEE Computer Society, pp. 385–391.

[66] LEE, U., ZHOU, B., GERLA, M., MAGISTRETTI, E., BELLAVISTA, P., AND CORRADI, A. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *Wireless Communications, IEEE 13*, 5 (october 2006), 52 –57.

[67] LEIDNER, J. L. Toponym resolution in text (abstract only): "which sheffield is it?". In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2004), SIGIR '04, ACM, pp. 602–602.

[68] LIEBERMAN, M. D., AND SAMET, H. Multifaceted toponym recognition for streaming news. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (New York, NY, USA, 2011), SIGIR '11, ACM, pp. 843–852.

[69] LIEBERMAN, M. D., SAMET, H., SANKARANARAYANAN, J., AND SPERLING, J. Steward: architecture of a spatio-textual search engine. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems* (New York, NY, USA, 2007), GIS '07, ACM, pp. 25:1–25:8.

[70] LIU, S., LIU, Y., NI, L. M., FAN, J., AND LI, M. Towards mobility-based clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2010), KDD '10, ACM, pp. 919–928.

[71] LIU, Z., JOY, A., AND THOMPSON, R. A dynamic trust model for mobile ad hoc networks. In *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of* (may 2004), pp. 80 – 85.

[72] LOZANO, A., AND STORCHI, G. Shortest viable path algorithm in multimodal networks. *Transportation Research Part A: Policy and Practice 35*, 3 (2001), 225 – 241.

[73] LU, J.-L., YEH, M.-Y., HSU, Y.-C., YANG, S.-N., GAN, C.-H., AND CHEN, M.-S. Operating electric taxi fleets: A new dispatching strategy with charging plans. In *Electric Vehicle Conference (IEVC), 2012 IEEE International* (march 2012), pp. 1 –8.

[74] MA, S., AND WOLFSON, O. Analysis and Evaluation of the Slugging Form of Ridesharing. *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2013).

[75] MA, S., WOLFSON, O., AND LIN, J. Iip: an event-based platform for its applications. In *Proceedings of the Second International Workshop on Computational Transportation Science* (New York, NY, USA, 2010), ACM, pp. 1–6.

[76] MA, S., WOLFSON, O., AND LIN, J. A survey on trust management for intelligent transportation system. In *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science* (New York, NY, USA, 2011), ACM, pp. 18–23.

[77] MA, S., ZHENG, Y., AND WOLFSON, O. T-share: A large-scale dynamic ridesharing service. In *Proceedings of the 29th IEEE International Conference on Data Engineering* (2013).

[78] MA, S., ZHENG, Y., AND WOLFSON, O. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering 99*, PrePrints (2014), 1.

[79] MASNADI-SHIRAZI, H., AND VASCONCELOS, N. Cost-sensitive boosting. *IEEE transactions on pattern analysis and machine intelligence 33*, 2 (Feb. 2011), 294–309.

[80] MATHUR, S., AND JIN, T. Parknet: drive-by sensing of road-side parking statistics. *Proceedings of the 8th international conference on Mobile systems, applications, and services* (2010).

[81] MEIER, R., AND CAHILL, V. Steam: Event-based middleware for wireless ad hoc networks. vol. 0, IEEE Computer Society, p. 639.

[82] MILUZZO, E., LANE, N. D., FODOR, K., PETERSON, R., LU, H., MUSOLESI, M., EISENMAN, S. B., ZHENG, X., AND CAMPBELL, A. T. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (New York, NY, USA, 2008), SenSys '08, ACM, pp. 337–350.

[83] MUN, M., ESTRIN, D., BURKE, J., AND HANSEN, M. Parsimonious mobility classification using GSM and WiFi traces. *Proceedings of the 5th Workshop on Embedded Networked Sensors* (2008), 1–5.

[84] NAWAZ, S., EFSTRATIOU, C., AND MASCOLO, C. ParkSense: A Smartphone Based Sensing System For On-Street Parking. *In Proceedings of the 19th ACM International Conference on Mobile Computing and Networking (MOBICOM 2013)* (2013).

[85] OUKSEL, A., AND LUNDQUIST, D. Demand-driven publish/subscribe in mobile environments. *Wireless Networks 16* (2010), 2237–2261.

[86] P. D'OREY, R. FERNANDES, M. F. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *IEEE Conf. on Intelligent Transportation Systems* (sept. 2010), vol. 1, pp. 1–6.

[87] PARRAGH, S. N., DOERNER, K. F., AND HARTL, R. F. Variable neighborhood search for the dial-a-ride problem. *Comput. Oper. Res. 37*, 6 (June 2010), 1129–1138.

[88] PATWARDHAN, A., JOSHI, A., FININ, T., AND YESHA, Y. A data intensive reputation management scheme for vehicular ad hoc networks. In *Mobile and Ubiquitous Systems: Networking Services, 2006 Third Annual International Conference on* (2006), pp. 1 –8.

[89] PERICH, F., JOSHI, A., FININ, T., AND YESHA, Y. On data management in pervasive computing environments. *IEEE Transactions on Knowledge and Data Engineering 16* (2004), 621–634.

[90] PO-YU CHEN, JE-WEI LIU, W.-T. C. A fuel-saving and pollution-reducing dynamic taxi-sharing protocol in vanets. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd* (sept. 2010), pp. 1–5.

[91] RABABAAH, A. Event Detection, Classification And Fusion For Non-Stationary Vehicular Acoustic Signals. *International Journal of Science of Informatics 1*, 1 (2011), 9–20.

[92] RASMUSSON, L., AND JANSON, S. Simulated social control for secure internet commerce. ACM Press, pp. 18–26.

[93] RAVI, N., DANDEKAR, N., MYSORE, P., AND LITTMAN, M. Activity recognition from accelerometer data. *AAAI* (2005).

[94] RAYA, M., AND HUBAUX, J.-P. Securing vehicular ad hoc networks. vol. 15, IOS Press, pp. 39–68.

[95] RAYA, M., PAPADIMITRATOS, P., GLIGOR, V., AND HUBAUX, J.-P. On data-centric trust establishment in ephemeral ad hoc networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE* (2008), pp. 1238 –1246.

[96] REDDY, S., MUN, M., BURKE, J., AND ESTRIN, D. Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks (TOSN) 6*, 2 (Feb. 2010), 1–27.

[97] RESNICK, P., AND ZECKHAUSER, R. *Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System*. Elsevier Science, Nov. 2002.

[98] REZENDE, C. G., ROCHA, B. P. S., AND LOUREIRO, A. A. F. Publish/subscribe architecture for mobile ad hoc networks. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing* (New York, NY, USA, 2008), ACM, pp. 1913–1917.

[99] SAILHAN, F., AND ISSARNY, V. Energy-aware web caching for mobile terminals. In *in Proceedings of the CDCS Workshop on Web Caching Systems* (2002), pp. http://www–rocq.inri.

[100] SANTANI, D., BALAN, R. K., AND WOODARD, C. J. Spatio-temporal efficiency in a taxi dispatch system. Tech. rep., School of Information Systems, Singapore Management University, October 2007.

[101] SAVELSBERGH, M. W. P. Local search in routing problems with time windows. *Annals of Operations Research 4* (1985), 285–305.

[102] SHOUP, D. *The High Cost of Free Parking*. American Planning Association, 2005.

[103] SIVAHARAN, T., BLAIR, G., AND COULSON, G. Green: A configurable and re-configurable publish-subscribe middleware for pervasive computing. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE* (2005), R. Meersman and Z. Tari, Eds., vol. 3760 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 732–749. 10.1007/11575771_46.

[104] STENNETH, L., WOLFSON, O., XU, B., AND YU, P. S. PhonePark: Street Parking Using Mobile Phones. *2012 IEEE 13th International Conference on Mobile Data Management* (July 2012), 278–279.

[105] STENNETH, L., WOLFSON, O., YU, P. S., XU, B., AND MORGAN, S. Transportation Mode Detection using Mobile Phones and GIS Information. *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2011).

[106] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2001), SIGCOMM '01, ACM, pp. 149–160.

[107] TAO, C.-C. Dynamic taxi-sharing service using intelligent transportation system technologies. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on* (sept. 2007), pp. 3209 –3212.

[108] TRIANTAFILLOU, P., AND AEKATERINIDIS, I. Content-based publish/- subscribe over structured p2p networks. In *In DEBS* (2004).

[109] TSUBOUCHI, K., HIEKATA, K., AND YAMATO, H. Scheduling algorithm for on-demand bus system. In *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on* (april 2009), pp. 189 –194.

[110] TULYAKOV, S., AND JAEGER, S. Review of classifier combination methods. *Machine Learning in Document Analysis and Recognition*, Figure 1 (2008), 1–26.

[111] UMAR MINHAS, JIE ZHANG, T. T., AND COHEN, R. Towards expanded trust management for agents in vehicular ad-hoc networks. vol. 5, International Journal of Computational Intelligence: Theory and Practice (IJCITP).

[112] VARRIALE, R., MA, S., AND WOLFSON, O. Vtis: A volunteered travelers information system. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science* (New York, NY, USA, 2013), IWCTS '13, ACM, pp. 13:13–13:18.

[113] WANG, Y., LIN, J., AND ANNAVARAM, M. A framework of energy efficient mobile sensing for automatic user state recognition. *Proceedings of the 7th international conference on Mobile systems, applications, and services* (2009).

[114] WEX, P., BREUER, J., HELD, A., LEINMULLER, T., AND DELGROSSI, L. Trust issues for vehicular ad hoc networks. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE* (May 2008), pp. 2800 –2804.

[115] WOLFSON, O., SISTLA, A. P., XU, B., ZHOU, J., CHAMBERLAIN, S., YESHA, Y., AND RISHE, N. Tracking moving objects using database technology in domino. In *Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems* (London, UK, UK, 1999), NGIT '99, Springer-Verlag, pp. 112–119.

[116] WOLFSON, O., XU, B., AND CHO, H. J. Multimedia traffic information in vehicular networks. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2009), GIS '09, ACM, pp. 480–483.

[117] WONG, K. I., BELL, AND MICHAEL, G. H. Solution of the dial-a-ride problem with multi-dimensional capacity constraints. *International Transactions in Operational Research 13*, 3 (May 2006), 195–208.

[118] XIANG, Z., CHU, C., AND CHEN, H. A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. *European Journal of Operational Research 174*, 2 (2006), 1117 – 1139.

[119] XU, B., WOLFSON, O., YANG, J., AND STENNETH, L. Real-time Street Parking Availability Estimation. *MDM 13: Proceedings of the 14th International Conference on Mobile Data Management*.

[120] YAMAMOTO, K., UESUGI, K., AND WATANABE, T. Adaptive routing of cruising taxis by mutual exchange of pathways. In *Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II* (Berlin, Heidelberg, 2008), KES '08, Springer-Verlag, pp. 559–566.

[121] YUAN, J., ZHENG, Y., XIE, X., AND SUN, G. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2011), KDD '11, ACM, pp. 316–324.

[122] YUAN, J., ZHENG, Y., ZHANG, C., XIE, W., XIE, X., SUN, G., AND HUANG, Y. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International*

Conference on Advances in Geographic Information Systems (New York, NY, USA, 2010), GIS '10, ACM, pp. 99–108.

[123] YUAN, J., ZHENG, Y., ZHANG, C., XIE, X., AND SUN, G.-Z. An interactive-voting based map matching algorithm. In *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management* (Washington, DC, USA, 2010), MDM '10, IEEE Computer Society, pp. 43–52.

[124] YUAN, J., ZHENG, Y., ZHANG, L., XIE, X., AND SUN, G. Where to find my next passenger. In *Proceedings of the 13th international conference on Ubiquitous computing* (New York, NY, USA, 2011), UbiComp '11, ACM, pp. 109–118.

[125] ZHANG, L., TIWANA, B., QIAN, Z., AND WANG, Z. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (2010).

[126] ZHANG, Y., HULL, B., BALAKRISHNAN, H., AND MADDEN, S. ICEDB: Intermittently-Connected Continuous Query Processing. In *International Conference on Data Engineering (ICDE)* (Istanbul, Turkey, April 2007).

[127] ZHANG, Y., ZHAO, J., AND CAO, G. Roadcast: a popularity aware content sharing scheme in vanets. In *In IEEE ICDCS* (2009), pp. 223–230.

[128] ZHAO, Y., STURMAN, D., AND BHOLA, S. Subscription propagation in highly-available publish/subscribe middleware. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware* (New York, NY, USA, 2004), Middleware '04, Springer-Verlag New York, Inc., pp. 274–293.

[129] ZHENG, Y., LI, Q., CHEN, Y., XIE, X., AND MA, W.-Y. Understanding Mobility Based on GPS Data. *Proceedings of the 10th international conference on Ubiquitous computing*, 49 (2008).

[130] ZHENG, Y., LIU, L., WANG, L., AND XIE, X. Learning transportation mode from raw gps data for geographic applications on the web. *Proceedings of the 17th international conference on World Wide Web*, 49 (2008).

[131] ZHONG, S., CHEN, J., AND YANG, Y. Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies* (march-3 april 2003), vol. 3, pp. 1987 – 1997 vol.3.

VITA

SHUO MA

EDUCATION

University of Illinois at Chicago Summer 2014
Doctorate in Computer Science GPA: **4.0/4.0**

Beijing University of Posts and Telecommunications 09/2004-06/2008
Bachelor in Computer Science GPA: **88/100**

RESEARCH EXPERIENCE

Here.com, A Nokia Business 12/2013-05/2014
Research Intern

- Big data analysis: investigate the correlation between the quality of predicated traffic and the amount of GPS probe data that are used to predict traffic

Microsoft Research Asia Beijing, China 05/2012-12/2012
Research Intern

- Developed a real-time city-scale taxi ridesharing system. The core of the system includes an index structure which facilitates fast searching for taxi candidates and a route scheduling algorithm. The output of this project includes a publication at ICDE'13 which won the Best Paper Runner-up Award and a demo that is developed using *Microsoft Silverlight* and *WCF*.

Department of Computer Science, University of Illinois at Chicago 08/2008-Present
Research/Teaching Assistant

- Developed an Android App that automatically detects parking/unparking activities using smartphone sensors
- Analyzed and evaluated the slugging form of ridesharing
- Developed a crowd-sourcing Twitter-based real-time travel information notification application. As a result, an *Android* APP named Volunteer Travel Information (VTI) was published to the Google Play platform. For more information about VTI, please visit http://cs.uic.edu/~sma/VTI.
- Developed a general information platform for Intelligent Transportation System applications.
- TAed a broad range of courses such as programming, data structures, algorithms, automata, software engineering, databases (please see here for a comprehensive list). Responsibilities include leading lab sessions, lecturing discussion classes, designing homework, designing exams and quizzes, grading, etc.

Department of Urban Planning and Policy, University of Illinois at Chicago 05/2013-12/2013
Research Assistant

- Developed a Javascript powered interactive map application which visualizes the environmental impact of railways to the neighboring areas.

Beijing University of Posts and Telecommunications Beijing, China 09/2007-06/2008
State Key Laboratory of Network and Technology
Research Assistant

- Performed *unit testing* for a software which automatically detects bugs in *Java* programs.
- Developed a service management platform for the broadcasting and television system using *LAMP* architecture.

AWARDS

Best Paper Runner-up Award, *29th IEEE International Conference on Data Engineering* 04/2013

NSF Travel Grant Award, *ACM SIGSPATIAL'13* 09/2013

Outstanding Teaching Assistant Award, *Dept. of Computer Science, UIC* 06/2010

First Class Scholarship, *Beijing University of Post and Telecommunications* 2004-2008

First Class Prize of National Collegiate Physics Competition 12/2005

SOFTWARE

- UPDector (Android App): Sensing Parking/Unparking activities using smartphones
- Volunteer Traveler Information System (Android App) http://www.cs.uic.edu/~sma/VTI/
- Taxi Ridesharing Simulator http://www.cs.uic.edu/~sma/ridesharing/

PUBLICATIONS

- **Ma, S.**, Wolfson, O., Xu, B. "UPDetector: Sensing Unparking/Parking Activities Using Smartphones". Submitted to *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2014.*
- **Ma, S.**, Zheng, Y., and Wolfson, O. "Real-time City-Scale Taxi Ridesharing". Accepted, *IEEE Transactions on Knowledge and Data Engineering*.
- **Ma, S.**, Wolfson, O. "Analysis and Evaluation of the Slugging Form of Ridesharing". *Proceedings of the 21th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2013. (acceptance rate: 39/228=17.0%)
- Varriale, R., **Ma, S**. Wolfson, O. "VTIS: A Volunteered Travelers Information System*"*. *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science* , *IWCTS '13*
- **Ma, S.**, Zheng, Y., and Wolfson, O. "T-share: A large-scale dynamic ridesharing service". *Proceedings of the 29th IEEE International Conference on Data Engineering (2013), ICDE'13*. **Best Paper Runner-up Award** (3 out of 460 submissions).
- **Ma, S.**, Wolfson, O., and Lin, J. "A survey on trust management for intelligent transportation system". *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science* (New York, NY, USA, 2011), *IWCTS '11*, ACM, pp.18–23.

- **Ma, S.**, Wolfson, O., and Lin, J. “IIP: an event-based platform for its applications”. *Proceedings of the Second International Workshop on Computational Transportation Science* (New York, NY, USA, 2010), *IWCTS ’10*, ACM, pp.1–6.

APPENDIX A: Rightslink Terms and Conditions for ACM Material

1. The publisher of this copyrighted material is Association for Computing Machinery, Inc. (ACM). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at).

2. ACM reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

3. ACM hereby grants to licensee a non-exclusive license to use or republish this ACM-copyrighted material* in secondary works (especially for commercial distribution) with the stipulation that consent of the lead author has been obtained independently. Unless otherwise stipulated in a license, grants are for one-time use in a single edition of the work, only with a maximum distribution equal to the number that you identified in the licensing process. Any additional form of republication must be specified according to the terms included at the time of licensing.

*Please note that ACM cannot grant republication or distribution licenses for embedded third-party material. You must confirm the ownership of figures, drawings and artwork prior to use.

4. Any form of republication or redistribution must be used within 180 days from the date stated on the license and any electronic posting is limited to a period of six months unless an extended term is selected during the licensing process. Separate subsidiary and subsequent republication licenses must be purchased to redistribute copyrighted material on an extranet. These licenses may be exercised anywhere in the world.

5. Licensee may not alter or modify the material in any manner (except that you may use, within the scope of the license granted, one or more excerpts from the copyrighted material, provided that

the process of excerpting does not alter the meaning of the material or in any way reflect negatively on the publisher or any writer of the material).

6. Licensee must include the following copyright and permission notice in connection with any reproduction of the licensed material: "[Citation] © YEAR Association for Computing Machinery, Inc. Reprinted by permission." Include the article DOI as a link to the definitive version in the ACM Digital Library. Example: Charles, L. "How to Improve Digital Rights Management," Communications of the ACM, Vol. 51:12, © 2008 ACM, Inc. http://doi.acm.org/10.1145/nnnnnn.nnnnnn (where nnnnnn.nnnnnn is replaced by the actual number).

7. Translation of the material in any language requires an explicit license identified during the licensing process. Due to the error-prone nature of language translations, Licensee must include the following copyright and permission notice and disclaimer in connection with any reproduction of the licensed material in translation: "This translation is a derivative of ACM-copyrighted material. ACM did not prepare this translation and does not guarantee that it is an accurate copy of the originally published work. The original intellectual property contained in this work remains the property of ACM."

8. You may exercise the rights licensed immediately upon issuance of the license at the end of the licensing transaction, provided that you have disclosed complete and accurate details of your proposed use. No license is finally effective unless and until full payment is received from you (either by CCC or ACM) as provided in CCC's Billing and Payment terms and conditions.

9. If full payment is not received within 90 days from the grant of license transaction, then any license preliminarily granted shall be deemed automatically revoked and shall be void as if never granted. Further, in the event that you breach any of these terms and conditions or any of CCC's Billing and Payment terms and conditions, the license is automatically revoked and shall be void as if never granted.

10. Use of materials as described in a revoked license, as well as any use of the materials beyond the scope of an unrevoked license, may constitute copyright infringement and publisher reserves the right to take any and all action to protect its copyright in the materials.

11. ACM makes no representations or warranties with respect to the licensed material and adopts on its own behalf the limitations and disclaimers established by CCC on its behalf in its Billing and Payment terms and conditions for this licensing transaction.

12. You hereby indemnify and agree to hold harmless ACM and CCC, and their respective officers, directors, employees and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.

13. This license is personal to the requestor and may not be sublicensed, assigned, or transferred by you to any other person without publisher's written permission.

14. This license may not be amended except in a writing signed by both parties (or, in the case of ACM, by CCC on its behalf).

15. ACM hereby objects to any terms contained in any purchase order, acknowledgment, check endorsement or other writing prepared by you, which terms are inconsistent with these terms and conditions or CCC's Billing and Payment terms and conditions. These terms and conditions, together with CCC's Billing and Payment terms and conditions (which are incorporated herein), comprise the entire agreement between you and ACM (and CCC) concerning this licensing transaction. In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall control.

16. This license transaction shall be governed by and construed in accordance with the laws of New York State. You hereby agree to submit to the jurisdiction of the federal and state courts located in New York for purposes of resolving any disputes that may arise in connection with this licensing transaction.

17. There are additional terms and conditions, established by Copyright Clearance Center, Inc. ("CCC") as the administrator of this licensing service that relate to billing and payment for licenses provided through this service. Those terms and conditions apply to each transaction as if they were restated here. As a user of this service, you agreed to those terms and conditions at the time that you established your account, and you may see them again at any time at http://myaccount.copyright.com

18. Thesis/Dissertation: This type of use requires only the minimum administrative fee. It is not a fee for permission. Further reuse of ACM content, by ProQuest/UMI or other document delivery providers, or in republication requires a separate permission license and fee. Commercial resellers of your dissertation containing this article must acquire a separate license.