M.Sc. Engineering Thesis

Activity-aware Ridesharing Group Trip Planning Queries for Flexible POIs

By

Mehnaz Tabassum Mahin

Student No.: 2015052020

Submitted to

Department of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Dhaka-1000, Bangladesh

March, 2018

Author's Contact

Mehnz Tabassum Mahin
Lecturer
Department of Computer Science & Engineering
Bangladesh University of Engineering and Technology.
Email: mehnaz_mahin@cse.buet.ac.bd

The thesis titled "Activity-aware Ridesharing Group Trip Planning Queries for Flexible POIs", submitted by Mehnaz Tabassum Mahin, Roll No. **1015052020**, Session October 2015, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on 15th March 2018.

Board of Examiners

1. Tanzima
Dr. Tanzima Hashem — Chairman (Supervisor)
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

2.
Dr. Md. Mostofa Akbar — Member (Ex-Officio)
Head and Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

3.
Dr. Md. Abul Kashem Mia — Member
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

4. Rifat
Dr. Rifat Shahriyar — Member
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

5. Nova Ahmed 15th March, 18
Dr. Nova Ahmed — Member (External)
Associate Professor
Department of Electrical and Computer Engineering
North South University, Bashundhara, Dhaka-1229.

Candidate's Declaration

This is hereby declared that the work titled "Activity-aware Ridesharing Group Trip Planning Queries for Flexible POIs" is the outcome of research carried out by me under the supervision of Dr. Tanzima Hashem, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Mahi 18/3/18

Mehnaz Tabassum Mahin

Candidate

Acknowledgments

First of all, I am grateful to the Almighty Allah for the good health and wellbeing that were necessary to complete my thesis work.

I would like to express my sincere gratitude to my supervisor Dr. Tanzima Hashem for her continuous support, patience, motivation and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. She shared her knowledge in interpreting subject topics and also valued my way of thinking to synthesize those topics. Her suggestions pushed me towards the direction of better thinking, her brilliant reviews refined me in working out my research problems, and her support gave me spirit to continue the work. I could not have imagined having a better supervisor and mentor for research work. I am extremely thankful and indebted to her for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Dr. Md. Mostofa Akbar, Dr. Md. Abul Kashem Mia, Dr. Rifat Shahriyar and specially the extrenal member Dr. Nova Ahmed.

I would like to express my gratefulness to my parents for their unceasin encouragement, never-ending support and attention. I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lemt their hand in this thesis work.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Ridesharing has become a popular model that enables users to share their rides with others in recent years. Traditional ridesharing services arrange ridesharing trips to travel between a fixed source and a fixed destination locations. Users need to visit point of interests (POIs) such as a supermarket or a pharmacy for performing various daily activities while traveling between fixed locations like an office and a home. In current ridesharing services, there is no guarantee that a user gets ridesharing options for the complete trip, e.g., for visiting from the office to a POI and then from a POI to the home. Again, the flexibility in visiting a POI a little bit far away instead of visiting the nearest POI with respect to fixed locations may increase the probability in getting ridesharing services. In this thesis, we introduce a novel type of ridesharing query, an *Activity-aware Ridesharing Group Trip Planning (ARGTP) query* that exhibit three novel features: (i) ensures a complete trip for visiting more than two locations, (ii) allows to visit both fixed and flexible locations, and (iii) provides true ridesharing services instead of a taxi like ridesourcing services by matching a group of riders' flexible trips with a driver's fixed trip. An ARGTP query considers the spatial proximity of the trips of the riders with that of a driver, and returns an optimal ridesharing group that minimizes the total cost of the ridesharing group. We develop the first solution to process ARGTP queries in real time. The efficiency of the ARGTP query processing algorithms depends on the number of candidate riders and the number of POIs to be explored. We introduce novel pruning techniques to prune the riders and refine the POI search space. We perform extensive experiments using both real and synthetic datasets to validate the efficiency and effectiveness of our approach, and show that our approach outperforms a baseline approach with a large margin.

Chapter 1

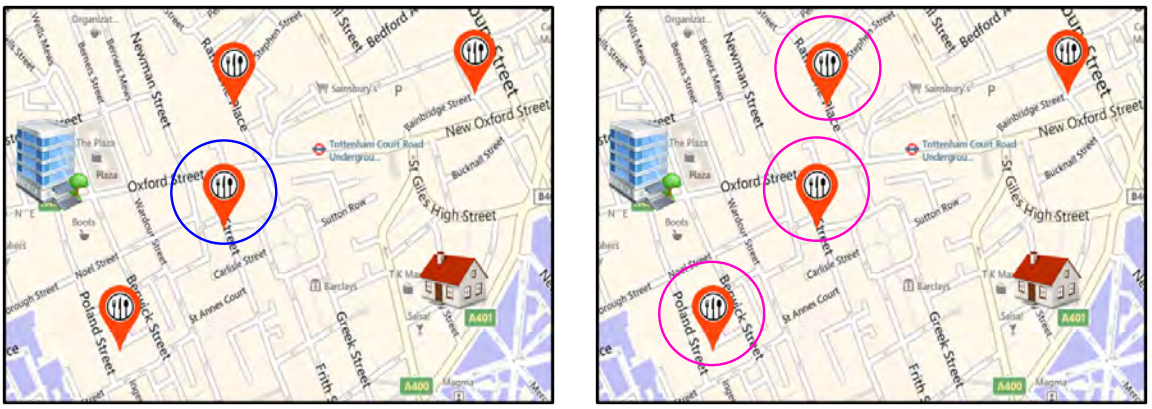
Introduction

With the proliferation of smart phones and GPS-enabled devices, ridesharing has become a popular and promising model that enables users to share their rides with others. Ridesharing combines independent trips of drivers and riders into a single one. Thus, ridesharing can reduce the road traffic and save energy, and have huge impact in the environment. Effective usage of the ridesharing applications depends on the coordination among drivers and riders considering time constraints, distance constraints, and more importantly the places that need to be visited. In this thesis, we propose a novel model that provides a true ridesharing service instead of taxi like existing ridesourcing applications such as Uber[1] and Lyft[2] by matching riders with a driver's predefined trip.

Traditional ridesharing services [1, 2, 3, 4, 5] are arranged considering only two fixed locations given by the users. To perform various daily activities like buying groceries and withdrawing money, a rider may want to visit a point of interest (POI) like a supershop or an ATM booth in between two fixed locations (i.e., source and destination). Thus, the rider needs two individual ridesharing trips to travel from source to the POI and then from the POI to the destination. However, there is no guarantee that the rider will get the second ridesharing trip after reaching the POI. In this thesis, we focus on arranging complete ridesharing trip to travel from the source to the destination via a POI.

[1] www.uber.com
[2] www.lyft.com

Usually a rider prefers to visit a POI that is the nearest with respect to her source and destination locations. A recent work [6] has shown that the flexibility in selecting the POI (e.g., any branch of a bank) increases the probability of getting a ridesharing trip. A rider may be happy to visit a POI that is a little bit far away from the nearest one if in return it increases the probability of getting a suitable ridesharing trip. Traditional ridesharing services consider neither such flexibility in selecting POIs nor a complete ridesharing trip including more than two locations. In this thesis, we introduce a new type of ridesharing query that considers riders' flexibility in selecting POIs, ensures a complete ridesharing group trip, and provides a true ridesharing service instead of a taxi like ridesourcing service by matching a group of riders' flexible trips with a driver's predefined trip. We call this query as *Activity-aware Ridesharing Group Trip Planning (ARGTP) query*. Ridesharing services can be dynamic or static. Dynamic ridesharing trips are arranged on a very short notice, while static trips are known in advance, at least few minutes before the departure time. Our focus is on static ridesharing services and we propose the first solution for efficient processing of ARGTP queries.



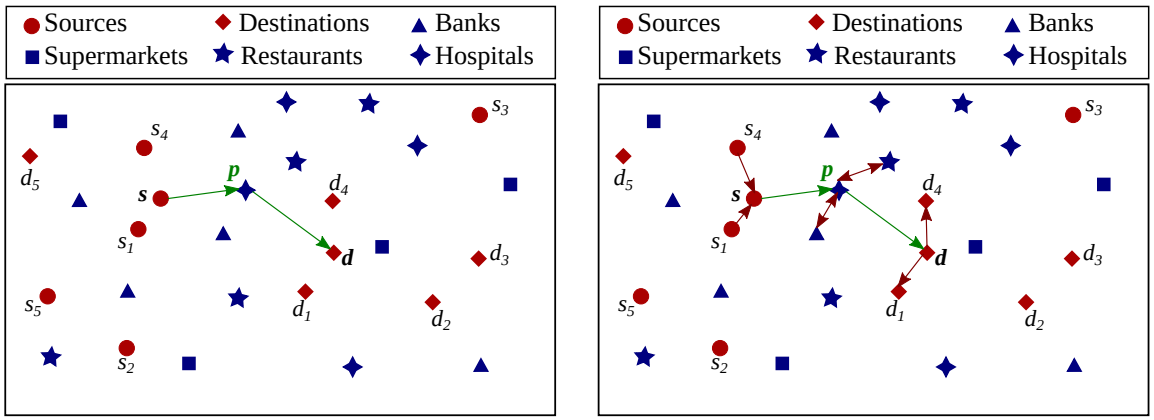
(a) A fixed POI (i.e., the nearest POI on the way from source to destination)

(b) Flexible POIs (i.e., POIs located within the maximum allowed distance)

Figure 1.1: A rider's flexibility in selecting an intermediary POI (e.g., restaurant) on the way from source (e.g., office) to destination (e.g., home)

1.1 ARGTP Queries

Taxi-like ridesourcing services assume that drivers do not have any planned trips for themselves and they are available for serving the riders, whereas in our ridesharing system, a driver is willing to share her vehicle with a group of riders only at a particular time when the driver has a planned trip. The driver's trip starts from a source location, goes through a POI, and ends at a destination location. On the other hand, a rider is willing to get a complete ridesharing trip for visiting a specific POI type. A rider can be flexible in visiting any branch of the required POI type within a distance limit instead of the nearest one. For example, in Figure 1.1(a), the circled POI is the nearest one with respect to a rider's source and destination locations, and in Figure 1.1(b), the circled POIs are located within the rider's maximum allowed distance.



(a) An ARGTP query takes a driver's trip and a set of riders' trips as input

(b) The optimal ridesharing group trip including riders r_1 and r_4

Figure 1.2: An example of an ARGTP query for $k = 3$

Riders are picked up and dropped off at a driver's source and destination locations. The group trip also halts for a specified period at the driver's POI location. The drivers whose source, destination and POI locations are within a rider's specified slugging distance limit from the rider's source, destination and POI locations, respectively are eligible to share a vehicle with the rider. Given a driver's trip and a set of riders' trips, An ARGTP query returns a ridesharing group that minimizes the total travel cost of group members and satisfies all spatial-temporal constraints specified by the members of the group. A group member's travel cost is the summation of the

distance of her source from the driver's source, the distance of the nearest required POI-type from the driver's fixed POI, and the distance of her destination from the driver's destination. Figure 1.2(a) shows trips of a driver and available riders, and Figure 1.2(b) shows the optimal ridesharing group for the driver.

1.2 Research Challenges

Computing ridesharing groups in real time is an essential criteria for the success of the ridesharing applications. A major challenge for an ARGTP query is to identify the optimal group from a large set of riders. For example, let there be 1000 candidate riders in the system for an ARGTP query and the required ridesharing group size be 6. Then there are ${}^{1000}C_6 = 1.368e^{+15}$ number of possible groups of riders for the query. Thus there can be a huge number of possible groups to be considered for an ARGTP query. The efficiency of an ARGTP query processing technique depends on the number of the candidate ridesharing groups considered for identifying the optimal ridesharing group.

Another major challenge is to explore the required POIs for the riders from a huge POI database. For example, California Road Network dataset [7] has about 87635 POIs with 63 different POI-types. For each POI-type, there are on average 1300 POIs. If there are 6 types of unique POIs required by a group, then the number of candidate POI sets for an ARGTP query is $(1300) \times (1300) \times (1300) \times (1300) \times (1300) \times (1300) = (1300)^6 = 4.83e^{+18}$, a huge number of candidate POI sets. Considering all of the possible ridesharing groups and POI sets to find the optimal query answer would not be an efficient solution. Thus, the efficient processing of an ARGTP query depends on the refinement of the rider and the POI search space.

1.3 Solution Overview

The key ideas behind the efficiency of our approach to evaluate ARGTP queries are the rider and POI search space refinement techniques. We develop two techniques to identify the riders that are not eligible to be included in the optimal ridesharing group.

For our first technique, we propose a heuristic to compute the upper bound of the travel cost of an optimal ridesharing group. Using the upper bound, we determine the maximum distance limit, and show that any rider whose source or destination is not located within the maximum distance limit from the driver's source or destination, respectively, cannot be part of the ARGTP query answer. Thus, all such drivers are pruned and not considered for processing an ARGTP query. In our second technique, we compute the minimum cost of a ridesharing group containing a specific rider as the lower bound, and prune the rider if the lower bound exceeds the upper bound.

Pruning riders may in turn reduce the number of required POI types. Furthermore, we derive the maximum allowed distance for a POI to be located from the driver's fixed POI location using the upper bound. If the maximum slugging distance of the candidate riders is less than the derived maximum distance, then we consider the maximum slugging distance to bound the POI search space. Otherwise, we take the derived maximum distance to refine the POI search space and explore the POIs of required types located within this distance from the driver's fixed POI.

After refining the rider and POI search space, our approach computes the travel cost of each rider and identify the rider with the smallest travel costs for identifying the optimal ridesharing group for an ARGTP query.

1.4 Contributions

The contributions of this thesis are summarized as follows:

- We introduce and formulate ARGTP queries. To the best of our knowledge, we first address the ARGTP query.
- We develop the first approach to process ARGTP queries efficiently in real time.
- We develop techniques to prune riders and POIs and thus, improve the efficiency of our ARGTP algorithm.
- We present extensive experiments using both real and synthetic datasets, to validate the effectiveness and the efficiency of our algorithms.

1.5 Thesis Organization

The next chapters are organized as follows. Chapter 2 presents the problem formulation of an ARGTP query. Chapter 3 introduces the related works. Chapter 4 presents our proposed efficient approach along with necessary algorithms. In Chapter 5, we discuss a baseline approach for processing ARGTP queries. The extensive experimental evaluation is presented in Chapter 6. Finally, in Chapter 7 we conclude the thesis with future research challenges.

Chapter 2

Problem Formulation

In Section 2.1, we present preliminaries and formulate the proposed ridesharing group trip planning query, i.e., an *Activity-aware Ridesharing Group Trip Planning (ARGTP)* query. We also introduce the basic notations and measures used throughout the thesis. In Section 2.2, we give an overview of the system architecture to process the ARGTP queries.

2.1 Activity-aware Ridesharing Group Trip Planning (ARGTP) Queries

In our system, drivers and riders specify their trip information to a ridesharing service provider (RSP). A trip starts from a source location, then visits a point of interest (POI) to perform an activity and finally ends at a destination location. An ARGTP query is initiated when a driver's trip is sent to the RSP. An ARGTP query enables riders to have a complete ridesharing trip for visiting more than two locations and considers riders' flexibilities in selecting POIs. The ARGTP query determines an optimal ridesharing group from the available set of riders for the driver that minimizes the total group cost by considering the spatial proximity of the riders with respect to the driver in road networks. Thus an ARGTP query requires a driver's trip and a set of riders' trips. Formally a driver's trip and a rider's trip can be defined as follows:

Definition 1 (**Driver's trip**). *A driver's trip is defined by* $\langle s, d, k, st, p, ht \rangle$, *where* s *and* d *represent a source location and a destination location respectively of the trip,* k *the car capacity,* st *the trip starting time and* p *the fixed POI location of the trip and* ht *the halting time at* p.

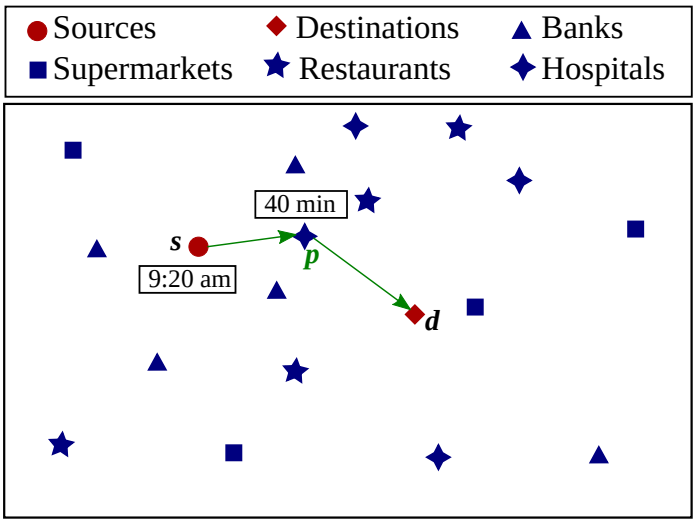


Figure 2.1: A driver's trip, $\langle s, d$, 4, 9:20 am, p, 40 minutes$\rangle$

Figure 2.1 shows an example of a driver's trip $\langle s, d$, 4, 9:20, p, 40 minutes$\rangle$. The driver starts the trip from s at 9:20, halts at a fixed POI p, e.g., a fixed hospital for 40 minutes, and ends the trip at d. The capacity of the driver's trip is 4.

Definition 2 (**Rider's trip**). *A trip of a rider* r_i *is defined by* $\langle s_i, d_i, t_i, ht_i, \lambda_i, sd_i, st_i \rangle$, *where the parameters have the following meanings:*

- s_i, d_i: *the source and the destination locations of* r_i
- t_i: *the POI-type that* r_i *wants to visit in between* s_i *and* d_i
- ht_i: *a duration interval that* r_i *wants to halt at the required* t_i*-type POI*
- λ_i: *a threshold distance limit that* r_i *can consider to travel more to get a ridesharing group trip*
- sd_i: *the maximum allowed slugging distance*
- st_i: *a time interval when* r_i *wants to be picked up at a driver's source*

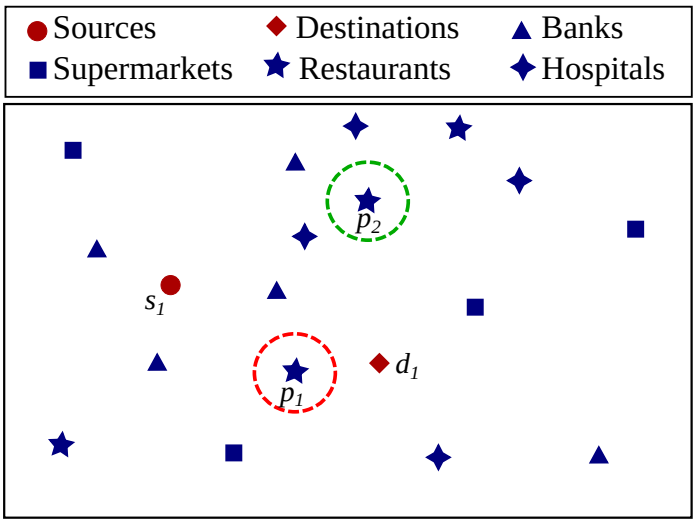
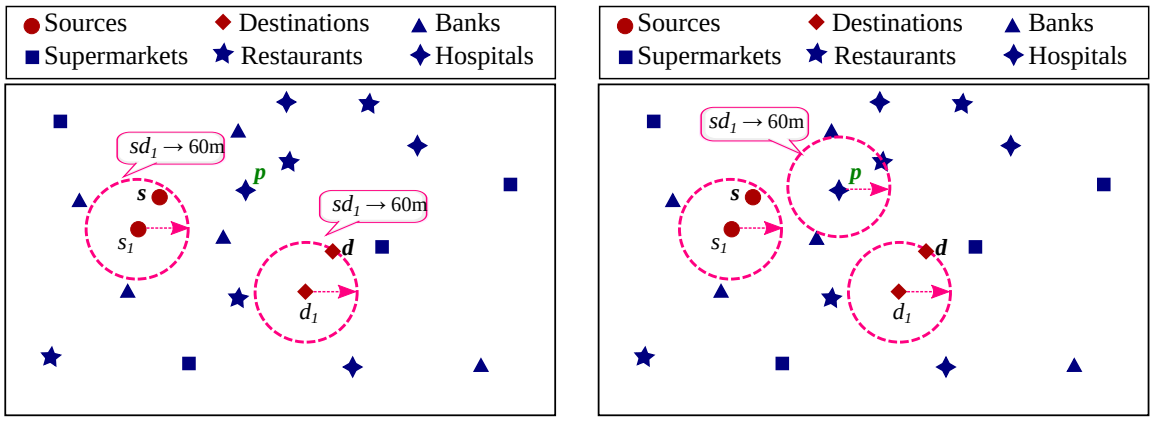


Figure 2.2: A rider r_1 can visit the restaurant p_2 instead of the nearest one p_1 on the way from s_1 to d_1 to get a ridersharing trip

Without loss of generality, an example of a rider's trip could be $\langle s_1, d_1$, restaurant, 30–40 minutes, 20%, 60 meters, 9:00– 9:30$\rangle$ (Figure 2.2). The rider wants to travel from s_1 to d_1 via a restaurant and also wants to stop at the restaurant for at least 30 minutes and at most 40 minutes. The rider is happy to travel any restaurant (e.g., p_2) that requires her to travel at most 20% (λ_i) more distance than the shortest trip via the nearest restaurant (i.e., p_1) with respect to s_1 and d_1. For ARGTP queries, we use *slugging* [1, 2, 8, 9], an effective model for sharing rides instead of ridesourcing like taxis. In the *slugging* model, the riders are picked up and dropped off at the driver's fixed locations. In the example, the slugging distance is 60 meters, i.e., the rider can travel at most 60 meters to be picked up and dropped off at the driver's fixed locations. The rider needs to start between 9:00–9:30 from a driver's source location. Next we elaborate the *slugging distance* and the *threshold distance* for a rider's trip.

Slugging Distance: Figure 2.3 shows the slugging distance of a rider's trip for the same example. Let there be a driver whose source, destination and the fixed POI are s, d and p respectively such that the driver's source s and destination d satisfy the slugging distance $sd_1 = 60$ m, i.e., s and d are located within 60 meters from the rider's source s_1 and destination d_1 respectively (Figure 2.3(a)). The rider can get



(a) The driver's source s and destination d satisfy $sd_1 = 60$ m

(b) A restaurant is located from the driver's fixed POI p within $sd_1 = 60$ m

Figure 2.3: The rider r_1 can get a driver's trip whose fixed locations are located within the slugging distance, $sd_1 = 60$ m

the driver's trip if there exists a restaurant located within 60 meters from the driver's fixed POI p (Figure 2.3(b)). Thus, the slugging distance sd_1 (= 60 m) states that the rider can travel at most 60 meters between (i) s and s_1, (ii) d and d_1, and (iii) p and the rider's POI.

In our system, a rider sets her slugging distance by considering that the rider has to arrive at the driver's source location within the driver's departure time, travel to the required POI and return back to the driver's fixed POI within the driver's halting time at the driver's fixed POI, and finally travel to her own destination from the driver's destination.

Threshold Distance: If a rider visits the nearest POI in between her source and destination locations, the trip is the shortest one with respect to her source and destination locations. Considering the fact, for ARGTP queries, it is assumed that to get a complete, suitable and cost-effective ridesharing group trip, a rider would be also happy to visit a POI a little bit far away but within a specified distance limit. We introduce this distance limit as *threshold distance*. In a rider's trip, the *threshold distance* is the maximum allowed distance that a rider can consider to travel more than her shortest trip to get the ridesharing group trip.

Let a rider r_i want to visit a POI of type t_i on the way from her source s_i to

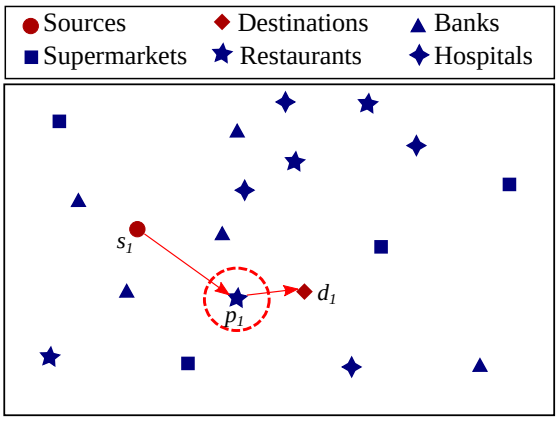
destination d_i, and the threshold distance be λ_i. Let there be a driver's trip from source s to destination d via a fixed POI p. If the shortest trip length from s_i to d_i via the nearest POI of type t_i is $(\ell_{t_i})_{shortest}$ and the maximum allowed ridesharing trip length is $(\ell_{t_i})_{max}$, then we can formulate the threshold distance as follows:

$$\lambda_i = \frac{(\ell_{t_i})_{max} - (\ell_{t_i})_{shortest}}{(\ell_{t_i})_{shortest}} \times 100\%$$

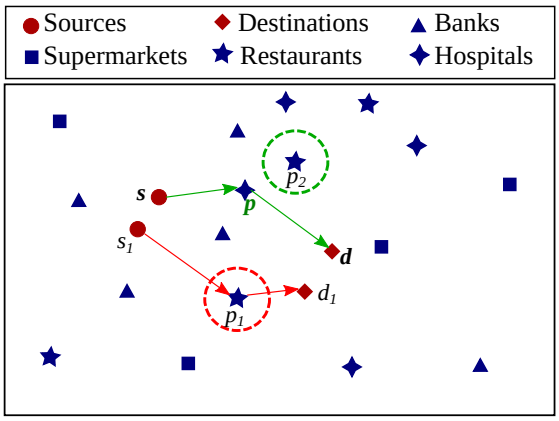
This implies that if the rider gets a ridesharing trip via a POI of type t_i, then the actual ridesharing trip length $(\ell_{t_i})_{actual}$ must be less then or equal to the maximum allowed ridesharing trip length $(\ell_{t_i})_{max}$, where $(\ell_{t_i})_{actual}$ is measured as the summation of the following distances: (i) the distance between s and s_i, (ii) the distance between s and p, (iii) twice of the distance between p and the rider's POI, (iv) the distance between p and d, and (v) the distance between d and d_i.

Figure 2.4 shows an example of the fact that how the threshold distance impacts on the ridesharing group selection. Let a rider r_1 wants to visit a restaurant on the way from source s_1 to destination d_1. If the rider takes an individual trip, then she may prefer to get the shortest trip via the nearest restaurant p_1. Figure 2.4(a) shows the shortest trip of the rider r_1. Now, let there be a driver whose source, destination and fixed POI locations are at s, d and p respectively. The restaurant p_2, which is the nearest one from p, satisfies the slugging distance constraint of the rider's trip. That's why, the rider r_1 can consider to visit the restaurant p_2 other than the nearest one p_1 (Figure 2.4(b)), if it also satisfies the threshold distance constraint. Figure 2.4(c) shows the complete ridesharing trip for the rider which satisfies the specified threshold distance, $\lambda_1 = 20\%$. For more clarification, Figure 2.4(d) shows an unacceptable ridesharing trip via a restaurant p_3 for the rider r_1 if she gets a driver's trip whose source, destination and fixed POI locations are at s', d' and p'. This trip requires more than 20% of the shortest trip, and violates the threshold distance constraint λ_1.

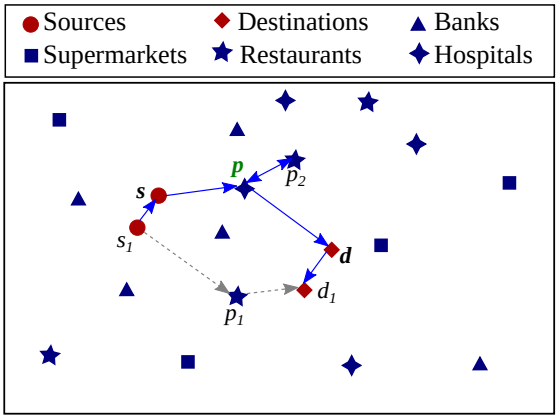
Let P be a set of POIs that are stored in a ridesharing service provider's (RSP's) database using an R^*-tree [61], p_{t_i} be a POI of type t_i, and R be a set of n_r riders in the system. If a driver's car capacity is k, there are $q = \binom{n_r}{k-1}$ possible ways to form ridesharing group of $k-1$ members from n_r riders. Let U be a set of such q ridesharing



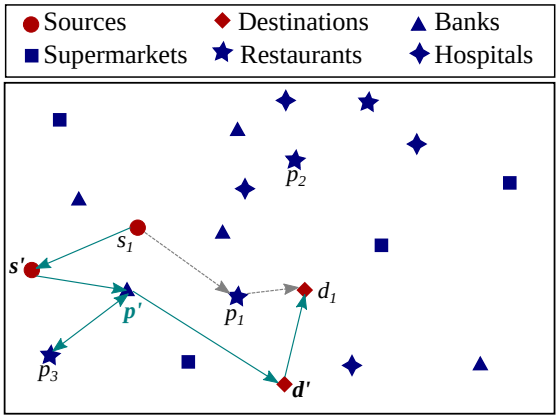
(a) The shortest trip for the rider r_1 via the nearest restaurant p_1 on the way from s_1 to d_1



(b) The rider r_1 can visit the restaurant p_2 instead of p_1 to get the driver's trip from s to d via p



(c) A complete and suitable ridesharing trip for the rider r_1 that satisfies the threshold distance, $\lambda_1 = 20\%$



(d) An unacceptable ridesharing trip for the rider r_1 that violates the threshold distance, $\lambda_1 = 20\%$

Figure 2.4: Determination of a complete and suitable ridesharing trip for a rider r_1 with threshold distance, $\lambda_1 = 20\%$

groups, where, $U = \{U_1, U_2, \ldots, U_q\}$. Formally, an *Activity-aware Ridesharing Group Trip Planning (ARGTP)* query is defined as follows:

Definition 3 (**ARGTP Queries**). *Given a set of POIs P, a set of n_r riders' trips R and a driver's trip with car capacity k, the set of possible ridesharing groups of $k-1$ riders U, an Activity-aware Ridesharing Group Trip Planning (ARGTP) query returns a ridesharing group $\Gamma \in U$ of $k-1$ riders and a set of POIs $\mathbb{P}$ required by the riders $r_i \in \Gamma$ such that the cost function C is minimized, i.e., $C \leq C'$ for any $\Gamma' \in U$ and $\Gamma \neq \Gamma'$.*

The cost function C is measured as follows:

Cost function C: All distances used in this thesis are assumed to be road network distances. We represent the road network with a graph $G(V, E, W)$, where a vertex $v \in V$ denotes a road junction, an edge $e \in E$ denotes a road between two vertices, and an weight $w \in W$ denotes the length of a road. Function $dist(.)$ returns the length of the shortest path between two locations in a road network. Suppose a rider r_i visits a POI $p_{t_i} \in P$ of type t_i in between s_i and d_i. If the rider participates in a ridesharing group trip, the rider herself first travels to s from s_i, then comes to p with the group, then again travels to her POI and returns back to p by herself, goes to d with the group, and finally travels to d_i by herself. Let the distance of the nearest POI of type t_i from p be d_{t_i}, where,

$$d_{t_i} = \min_{p_{t_i} \in P} \{dist(p, p_{t_i})\} \tag{2.1}$$

and let the travel cost of a rider r_i with respect to the source s_i and the destination d_i be χ_i, where,

$$\chi_i = dist(s_i, s) + dist(d, d_i). \tag{2.2}$$

The cost C_i of a rider r_i is defined as,

$$C_i = \chi_i + 2 \times d_{t_i}, \tag{2.3}$$

and the cost C for a ridesharing group Γ is defined as,

$$C = \sum_{r_i \in \Gamma} C_i. \tag{2.4}$$

Table 2.1 summarizes the notations we commonly used throughout the thesis:

Table 2.1: Notations and their meanings

Notations	**Meanings**
r_i	A rider's ID
P	A set of POIs in the road network
R	A set of riders present in the system
T	A set of POI-types required by the riders $r_i \in R$
$dist(u, v)$	The shortest path distance from u to v in the road network G
p_{t_i}	A desired POI of type t_i
d_{t_i}	The distance of the nearest POI of type t_i from the driver's fixed POI p
R_{t_i}	A set of riders whose desired POI-type is t_i
R_T	A set of riders-set R_{t_i} for each POI-type $t_i \in T$
Γ	The optimal ridesharing group
$\mathbb{P}$	The set of corresponding POIs of the optimal ridesharing group Γ
C	The optimal ridesharing group cost

2.2 System Overview

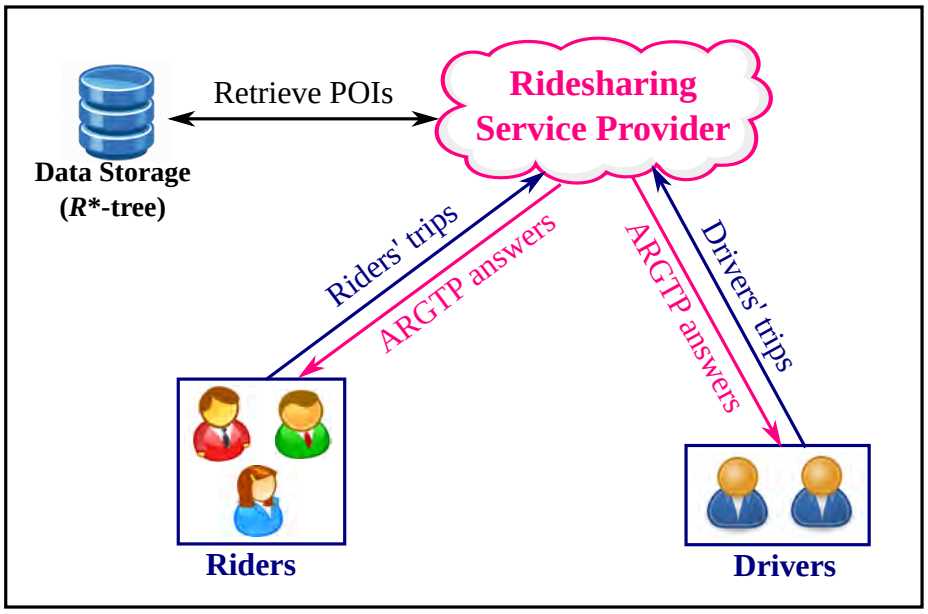


Figure 2.5: System architecture

Figure 2.5 shows an overview of the system architecture. The riders submit their required trips to a ridesharing service provider (RSP). Each rider provides the required POI-type along with source and destination locations. Each rider's trip also contains some constraints like slugging distance, threshold distance and time constraints defined by the rider. Similarly, the drivers send their trip information to the RSP. Each driver provides source, destination and fixed POI locations along with the car capacity. A driver's trip initiates an ARGTP query. The POI information is indexed using an R^*-tree [61] in the data storage of the RSP. Considering all the constraints given by the riders and the car capacity of the driver, the RSP processes the ARGTP query and returns the optimal ridesharing group trip information as the ARGTP answer to the riders and the driver included in the group.

Chapter 3

Related Works

In this chapter, we discuss the existing works related to our research problem. In Section 3.1, we discuss the group trip planning queries, where group members are predefined and the service provider schedules a group trip for them. In Section 3.2, we present different existing ridesharing models and their applications. There are some existing algorithms to provide efficient ridesharing services in the literature. We elaborately discuss the existing ridesharing algorithms in Section 3.3. Recently, researchers have shown their interest in some context based ridesharing services besides the traditional ones. In Section 3.4, we discuss the context based ridesharing queries addressed by the researchers.

3.1 Group Trip Planning Queries

Group trip planning (GTP) queries have been addressed in the literature [10, 11, 12, 13], where a set of POIs of different types are returned for a group that minimizes the travel distance with respect to the source and destination locations of group members. In [12], the authors addressed the GTP queries for Euclidean space. The proposed algorithm to process GTP queries is not scalable and efficient enough for road networks. Moreover, their appraoch uses independent R-trees [14] for each POI-types, which is impractical for road networks. Addressing these issues, Hashem *et. al* [11] proposed algorithms to process GTP queries efficiently in both Euclidean space and road networks. The authors also proposed optimal algorithms to evaluate kGTP

queries which find k sets of POIs that provide the k smallest aggregate trip distance with less computational overhead.

In [15], the authors introduced a Socio-Spatial Group Query (SSGQ) to select a group of nearby attendees with tight social relation. In [13], the authors proposed a new variant of a GTP query, subgroup trip planning queries, that return the optimal trips for different subgroup size. The authors in [16] introduced dynamic group trip planning queries (DGTP), where the groups change dynamically over the duration of trips and members are selected from a predefined group. Group members can dynamically join and leave the group trip at any point of the road network, and travel the same set of POIs together during the trip.

Thus, in a GTP query, a group is given and the same POIs are visited by the group members, whereas in an ARGTP query, a group is determined and different POIs are visited by the group members in between their source and destination locations.

3.2 Ridesharing Models

Ridesharing services has recently become popular among users and researchers [17, 18, 19, 20]. Ridesharing has different variants where the drivers can either require passenger trips to be inclusive (both of rider's source and destination are part of the driver's trip) or partial (any of source and destination or both fall outside the driver's trip). In addition, driver's trips can be unchanged, or can include detours to pickup passengers. The drivers might choose to pickup single or multiple passengers. The ridesharing problem is usually modeled as an optimization problem to minimize the travel cost [21] or to maximize the number of participants [22].

Furuhata *et. al* [23] discussed different ridesharing systems such as dynamic ridesharing [24, 25], static ridesharing [1, 4], carpooling [5], dial-a-ride (DARP) [3, 26]. The authors in [24] first addressed the dynamic taxi ridesharing problem for a large number of taxis. In a recent work [27], the authors proposed a new ridesharing model where a driver has an expectation rate which is to be satisfied by the shared route percentage between the driver and the rider.

3.2.1 Dynamic Ridesharing

Dynamic ridesharing is a service that matches up the riders with the drivers and provides shared vehicle rides in real time or on a short notice. Dynamic ridesharing services have been gaining attention in the research fields [20, 24, 28, 29, 30]. In dynamic ridesharing systems, both of the drivers and the riders are the independent users who can dynamically register for trips. Thus, the riders and the drivers can leave and enter the system at any point of the road, and so the number of drivers is continuously varying in the system. In addition, since both drivers and riders are independent, they can have individual preferences for the rides. Depending on their preferences, they can accept or decline the ridesharing offers. In [30], the authors presented a survey on optimization techniques for centralized dynamic ridesharing problems. The dynamic ridesharing problem that considers to match up multiple riders to be picked up for a driver's ride is NP-hard [19, 31].

Dial-a-ride problem (DARP) is a vehicle transport planning problem for planning to pickup and drop-off the passengers. The objective is to allocate riders' trip requests to the available vehicles such that they service all the riders according to their time windows and minimize the overall trip distance or the number of vehicles used. In dynamic DARP [24], the drivers are registered in a system, the vehicle requests are placed in real time, and the matched vehicles are allocated for the eligible riders, whereas, in dynamic ridesharing the drivers are not employed by any company and can decide greedily to pick up the riders considering their own privilages.

In [19], the authors proposed a dynamic ridesharing system where the drivers with similar routes are considered to form groups. A driver is selected from a group such that her shared ride provides others a cost-effective ride. In all of these dynamic ridesharing models, the groups are pre-determined and among the group members either a sub-group is chosen as riders or a group member is chosen as a driver. On the other hand, for ARGTP queries, ridesharing group is determined for a driver's trip such that the group travel cost is minimized. Thus, these types of dynamic ridesharing models are not suitable for processing ARGTP queries.

3.2.2 Static Ridesharing

In static ridesharing model, both of the driver's trip and the rider's trip are known in advance. Sometimes it can be achieved by tracking regular activities of users. Some popular static ridesharing models are: carpooling, static dial-a-ride (DARP), slugging etc. The main property for a static model is to know the trip information in advance, at least for a little bit earlier. Since according to our system, the trips of riders and drivers are known to the RSP, it can be categorized as a static ridesharing system.

Carpooling is a popular application of ridesharing static models, where the drivers need to change their routes to share their rides with the riders. They declare their availability for pick-up and then bring back riders later. Small size carpooling can be solved optimally by using linear programming techniques [32, 33]. To deal with large scale carpooling problems, the authors in [4, 34, 35] proposed heuristics algorithms.

Dial-a-ride problem (DARP) [3, 36] designs vehicle routes and schedules trips for users in between two fixed locations. If the users' rides are known in prior, then these types of problems are static DARP [19, 37, 38]. In [4, 39], the authors focused on the single vehicle problem where one vehicle is considered to identify the travel route, determine the ridesharing group and schedule riders' trips. Some online ridesharing services [40, 41] focus on the fast responses of the users without concerning about the optimal solution. Thus there is no guarantee to get a ridesharing trip between two locations. The large scale static DARP is known to be NP-hard [40]. For small number of vehicles, DARP can be solved exactly based on integer programming [42], whereas, heuristics are the most popular methods [3, 37, 39] for large scale DARP.

Slugging [2] is an effective and popular form of ridesharing where a rider walks to the origin of the driver's trip, boards at the driver's departure time, comes to the driver's destination along with the driver, and then travels to her own destination by herself. Slugging is the simplest form of ridesharing in the sense that there is no detour required for a driver, and the trip is cost-effective for both riders and the driver. In [1], Ma *et. al* studied slugging model from a computational perspective to improve the utilization of vehicles and to reduce the car congestion on the road. To model the system architecture for processing ARGTP queries, we consider this

slugging model where the riders are picked up and dropped off the driver's fixed locations. Our slugging model is different from the typical slugging model in the sense that the riders can travel to the driver's fixed locations by any means, they can walk, travel by vehicles, run and so on.

3.3 Ridesharing Algorithms

There exist efficient algorithms [17, 18, 34, 43, 44, 45] that can compute traditional ridesharing groups for visiting only two fixed locations. The authors in [17, 18] proposed efficient algorithms to match riders' trips to vehicles dynamically.

Traditionally, carpooling is a typical ridesharing model that forms a ridesharing group with a group of riders considering the fact that their travel routes and schedules are overlapped with each other [46]. The authors mentioned that carpooling is effective due to the reduced congestion, pollution and the monetary benefits for sharing the trips among the riders. If same group of riders share their rides with each other for a long time, then the arrangements become static carpooling. For carpooling, the user trajectories are used to identify the overlapping routes of the riders [47], and the groups are suggested accordingly. In [18], the authors proposed a collective transport system where the riders' trips are clustered to identify the cost-optimal ridesharing group who can travel collectively.

Some traditional taxi-sharing services [48, 49, 50, 51] usually send a taxi close to a passenger according to the passenger's request. They actually schedules a taxi-sourcing or ridesourcing trips in lieu of ridesharing. Ma *et. al* in [24] proposed a dynamic taxi ridesharing framework for GPS-equipped taxis in a city which serves a large number of taxis at a time and aims to reduce the total travel distance of these taxis. Ma *et. al* [52] proposed a taxi-sharing system using mobile-cloud architecture where the ride requests of taxi passengers are accepted in real time via smart devices, and passengers are picked up at their preferred locations. They consider the monetary issues to guarantee effective ridesharing services.

In [53], the authors proposed an R-tree based scalable ridesharing algorithm to schedule ridesharing trips where the drivers' trips are fixed and the riders' trips are

matched to the fixed trips of the drivers based on their social and economical preferences. In [33], the authors considered to incorporate riders' daily routine commutes for scheduling a ridesharing trip. In [54], the authors proved that the time window problems at the pickup and delivery time can be resolved.

However, these existing ridesharing services schedule ridesharing trips among riders and the driver for visiting two fixed locations. They do not consider user's flexibility in the selection of POIs. If a rider wants to visit any branch of a shopping mall on the way from office to home, then using traditional ridesharing services she has to manage two separate ridesharing trips. Sometimes it may happen that she may not get ridesharing options for both trips. Thus, we introduce a new type of ridesharing query, an ARGTP query, that enables riders to get a complete ridesharing trip for visiting more than two locations and considers riders' flexibilities in the selection of POIs.

3.4 Context Based Ridesharing Queries

Researchers have shown their interest to develop different context based algorithms to process ridesharing queries in real time. The context like activity, utility, community, social-awareness, privacy of users are prioritized among researchers.

In [35, 55], the authors addressed trust-conscious ridesharing and involved social networks to solve the problems. In [44, 56], the authors involved social networks to retrieve a group of riders where the riders' trips are similar to the driver's trip. Incorporating social networks data by the service provider raises serious threats to reveal personal data to the untrusted service provide. To address these issues, the authors in [57] introduced a novel technique to form ridesharing groups that reveals social data in community levels. Goel *et. al* [43] developed a privacy preserving ridesharing model where the riders can choose their ridesharing partners based on their preferences. In [58], the authors studied the effect of the riders' privacy and security by tracking their information precisely and also incorporating imprecision to design a ridesharing prototype. They focused on the privacy-based techniques to implement privacy preserving ridesharing.

In [59], the authors proposed a variant of DARP to schedule the optimal ridesharing trip that maximizes utilities like spatial-temporal and car capacity constraints. However, all of these are unable to handle users flexibility in selecting POI locations, and also the trips for visiting more than two locations. On the other hand, both fixed and flexible locations are considered in an ARGTP query. An ARGTP query also considers the spatial-temporal constraints provided by the riders and car capacity constraints. At the same time, ARGTP queries enable users to get complete ridesharing trips for visiting more than two locations.

Recently, it has been shown in [6] that the rate of finding a ridesharing trip between two POIs is improved by incorporating the flexibility in the selection of POIs. However, in [6] only trips between two locations (fixed or flexible) are considered, and no ridesharing group is formed for visiting more than two locations. Thus it is not guaranteed that if a rider goes to a POI other than the nearest one, she can manage a ridesharing trip for returning from the flexible POI to a fixed destination. In [60], the authors incorporated the flexibility such that a group of riders agree on a common destination from a possible set of destination locations. However, they do not consider to form a ridesharing group for visiting more than two locations. On the other hand, an ARGTP query enables users to get a complete ridesharing group trip via intermediary POIs for both fixed and flexible locations.

Chapter 4

Efficient Approach

A ridesharing service provider (RSP) has to explore a large set of riders and a huge database of point-of-interests (POIs) to identify the optimal ridesharing group as the answer of an *Activity-aware Ridesharing Group Trip Planning (ARGTP)* query. Considering all available riders and a huge POI database for identifying the query answer would not be an efficient solution. The efficiency of ARGTP queries depends on the size of the sets of the considered riders and POIs to find the optimal ridesharing group. We propose an efficient approach ARGTP_EA to process ARGTP queries. Our approach prunes the riders and POIs that cannot be part of the optimal answer and thereby refines the riders and the POI search space.

In Section 4.1, we give an overview of our efficient approach to process ARGTP queries. Section 4.2 presents the trivial pruning process of the riders by considering the spatial-temporal constraints of the specified trips. We discuss the steps of our efficient approach from Section 4.3 to Section 4.7 in details. Then we propose our algorithms to process ARGTP queries efficiently in Section 4.8. Finally, we present the correctness proofs for the algorithms in Section 4.9 and complexity analysis of the algorithms in Section 4.10.

4.1 Overview

In our system architecture, the riders and the drivers submit their trip information to the RSP. An ARGTP query is initiated as soon as a driver submits her trip to the RSP. Then the RSP identifies the set of riders' trips among the available riders' trips that minimizes the group cost for the driver's trip. Though the RSP can prune some riders based on their spatial and temporal constraints, still the remaining number of riders is large. Considering such a large set of riders and their required POI types for processing an ARGTP query incurs a high processing overhead.

We develop an efficient approach (ARGTP_EA) for processing ARGTP queries. The key idea behind the efficiency of our ARGTP_EA is to prune a significant number of riders and POIs that cannot be the part of the optimal query answer. Figure 4.1 shows the steps of ARGTP_EA.

ARGTP_EA first determines the upper bound of the optimal cost of the ridesharing groups using a heuristic technique. Section 4.3 describes the upper bound computation technique. Then the approach computes the lower bound of the optimal group cost if the group includes a specific rider. Section 4.4 elaborates the lower bound computation technique. In the next step, the approach prunes the riders in two ways: using the derived upper bound and the lower bounds (see Section 4.5). Then it prunes the POI-types based on the pruned riders and using the upper bound (see Section 4.6), and thus refines the POI search space. After that, ARGTP_EA explores the refined POI search space and search the nearest POIs for the remaining required POI-types with respect to the driver's fixed POI location. Again, the required POI types that are not found in the refined search space further prunes the riders who want to visit those POI types. Then with the reduced set of riders and retrieved POIs from the database, the approach computes the cost of each rider that has not been pruned for the ridesharing trip, and sorts the riders' list according to the calculated cost in an ascending order. Finally, ARGTP_EA forms the ridesharing group with the first $k-1$ riders from the list. This ridesharing group is the optimal one that minimizes the group cost function, which is returned as the query answer.

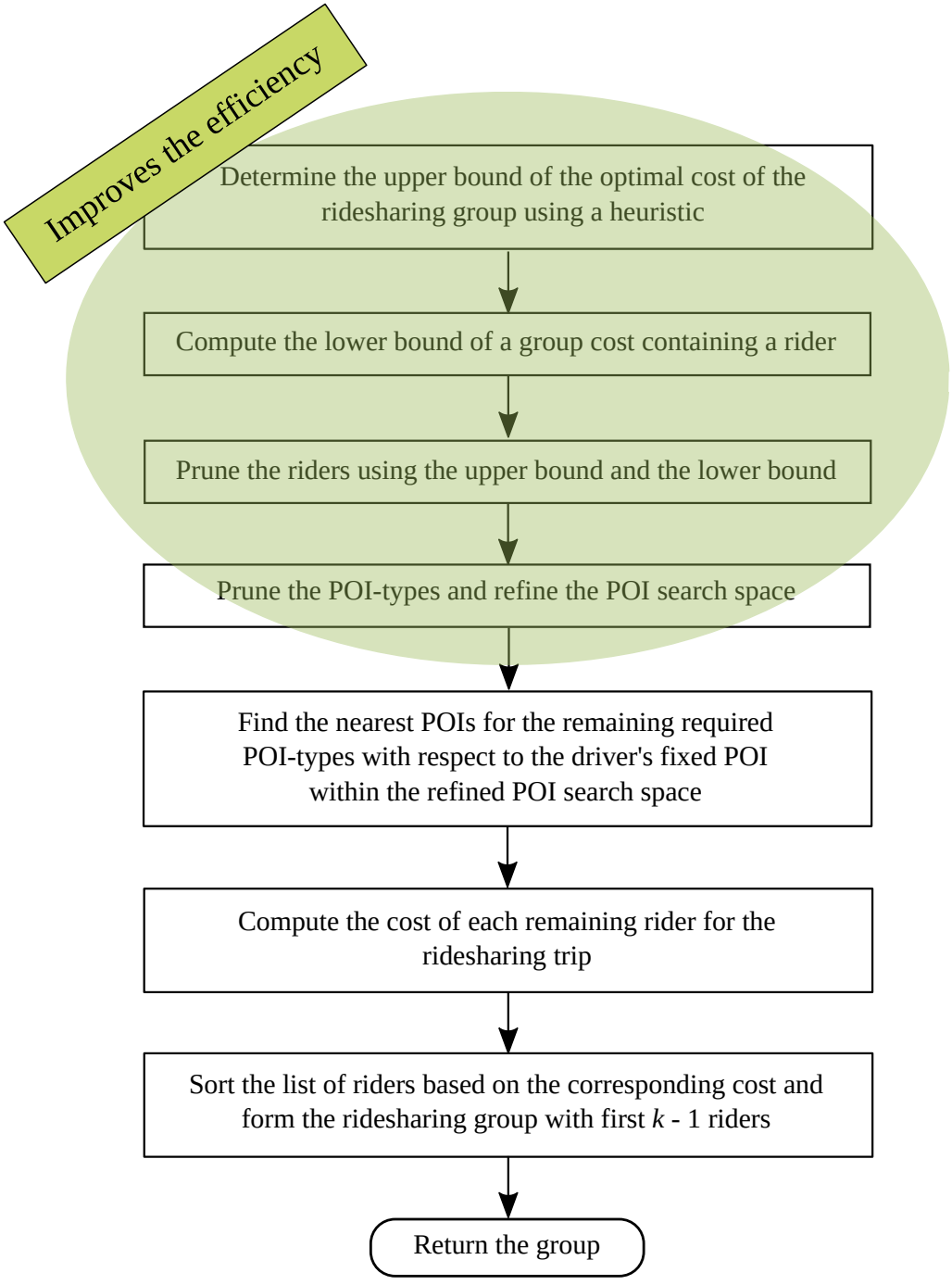


Figure 4.1: Overview of the ARGTP_EA approach

4.2 Trivial Pruning

Before processing the steps of the efficient approach (ARGTP_EA), the RSP applies the trivial pruning to reduce the available riders to form a ridesharing group. This trivial pruning phase is based on the constraints given by the riders and the driver. After this phase, the RSP determines the set of eligible riders for the ARGTP query. In this trivial pruning phase, a rider r_i is pruned if any of the following cases occurs:

1. The starting time st of the driver's trip exceeds the late bound of the rider r_i's start time interval $st_i.l$, i.e., $st > st_i.l$,

2. The halting time at the driver's fixed POI ht does not satisfy the halting time interval ht_i at the required POI type of a rider's trip, i.e., the early bound $ht_i.e > ht$ or the late bound $ht_i.l < ht$,

3. Slugging distance constraint sd_i of a rider r_i for any of the source or destination or both is violated, i.e., $dist\ (s_i,\ s) > sd_i$ or $dist\ (d,\ d_i) > sd_i$ or both of them.

4. Considering that the rider's POI and the driver's fixed POI are same, threshold distance constraint is violated, i.e., $dist(s, s_i) + dist(d, d_i) > (1 + \lambda_i) \times \ell_{shortest}$, where, λ_i represents the rider r_i's threshold distance and $\ell_{shortest}$ is the shortest trip length via the nearest POI with respect to s_i and d_i.

After performing the initial pruning, the RSP considers the remaining riders as the eligible riders to process ARGTP queries.

4.3 Upper Bound Computation

In this section, we propose a heuristic technique to find the upper bound of the optimal group cost for an ARGTP query. This upper bound allows to prune riders and POIs which cannot contribute in the optimal solution of an ARGTP query. If the car capacity for an ARGTP query is k, then excluding the driver a ridesharing group size n will be $k - 1$, i.e., $n = k - 1$. It may happen that more than one rider want to visit a POI of same POI-type. Let there be n' different number of POI-types those are required by n riders of a ridesharing group, where, $1 \leq n' \leq n$.

The proposed heuristic technique requires the following four steps:

Step 1: The technique retrieves n' number of nearest POI-types from the driver's fixed POI to form a ridesharing group of size n.

Step 2: Then the technique incrementally retrieves nearest POIs until at least $x\%$ of the required POI-types for all candidate riders have been retrieved.

Step 3: After that, the technique considers all the riders who want to visit those retrieved POIs and computes their cost for the ridesharing trip according to Equation (2.3).

Step 4: Finally, the technique calculates the upper bound by adding the lowest n costs of the considered riders in Step 3.

In the heuristic technique, we consider the fact that a rider who wants to visit the nearest POI from the driver's fixed POI p can have the source (or destination) far away from the driver's source (or destination). If we ignore the fact, then it may happen that the upper bound becomes a loose bound. We have observed during the experiments that if we retrieve only n' number of POI-types to compute the upper bound, then in most of the cases, some riders are selected whose either source or destination location is the furthest one from the driver's source or destination, and thus the upper bound tends to be loose. On the other hand, if we retrieve some additional POI-types during the upper bound computation, the bound tends to be tight for almost all of the cases. Thus to determine a tight upper bound of the group cost function, in Steps 1 and 2, we retrieve at least $x\%$ of the required POI-types for all candidate riders, where x is decided in experiments. Suppose that the required number of POI-types for all candidate riders is 30, and n' is 4. If $x = 20$, then we have to retrieve at least 20% of 30 = 6 POI types during the upper bound computation. Here, $n' < 6$, and thus, we retrieve nearest POIs for additional $6 - n' = 2$ POI types in Step 2. If $x = 10$, then we have to retrieve at least 10% of 30 = 3 POI-types, which is less than n'. In this case, we retrieve no POI in Step 2.

Figure 4.2 shows an example, where there is a driver's trip from source s to destination d via a fixed POI p, and the car capacity $k = 5$. Thus, for the ARGTP

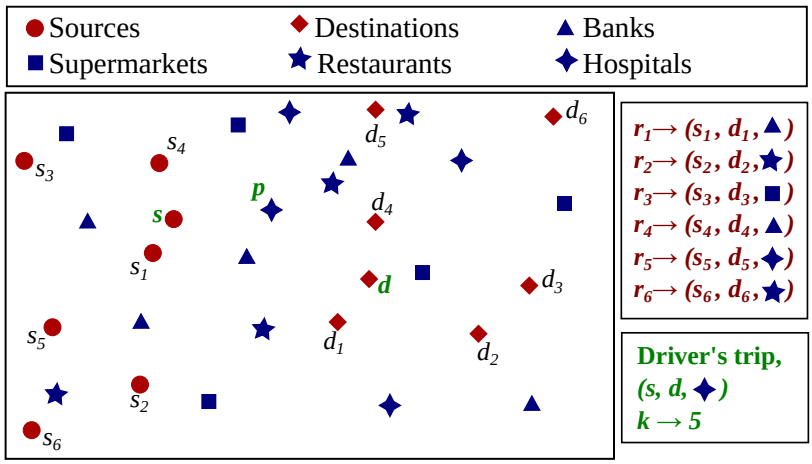
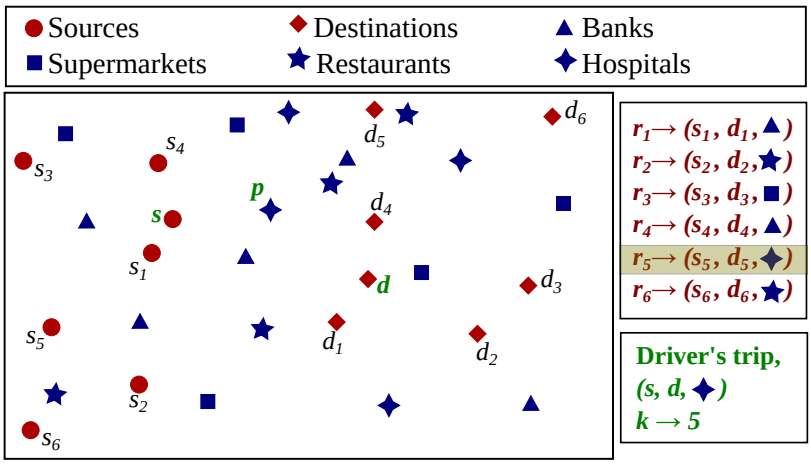


Figure 4.2: An example scenario for an ARGTP query with $k = 5$

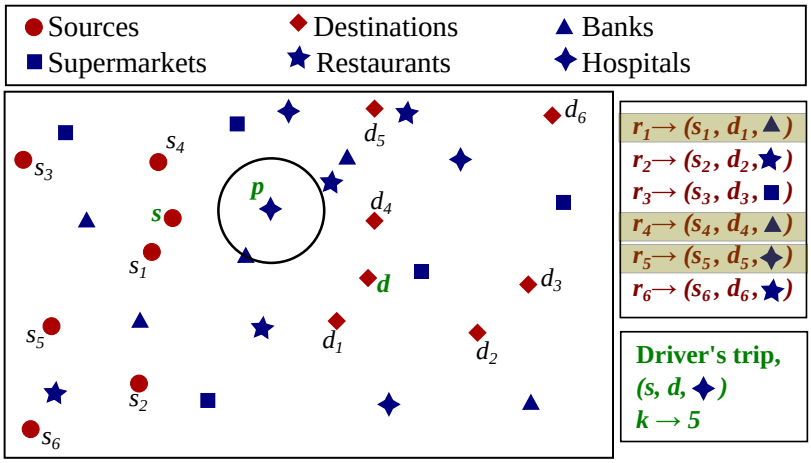
query, the ridesharing group size $n = 4$. There are 6 candidate riders whose source and destination pairs are (s_1, d_1), (s_2, d_2), (s_3, d_3), (s_4, d_4), (s_5, d_5) and (s_6, d_6), respectively. In between their source and destination locations, the riders r_1 and r_4 want to visit a bank, the riders r_2 and r_6 want to visit a restaurant, the rider r_3 wants to visit a supermarket, and the rider r_5 wants to visit a hospital.

Figure 4.3 shows the steps of the upper bound computation. At first, the hospital which is also the driver's fixed POI is considered and the rider r_5 is considered accordingly (Figure 4.3(a)). Then the next nearest POI, i.e., the nearest bank, is retrieved and both of the riders r_1 and r_4 are considered (Figure 4.3(b)). Since the nearest POIs for 2 POI types and 3 riders are considered so far, and the required ridesharing group size is 4, the technique continues to retrieve the nearest POIs from p. The next nearest POI, i.e., the nearest restaurant, is retrieved and both of the riders r_2 and r_6 are considered (Figure 4.3(c)). Using these 3 nearest POIs and corresponding 5 riders, the technique can form a ridesharing group of size 4. Here, the number of different required POI types $n' = 3$ for the group of size 4, and the required number of POI-types for all candidate riders = 4. If $x = 20$, then we have to retrieve at least 20% of $4 = 0.8 \approx 1$ POI-types. Since $n' > 1$, we stop the POI retrieval process for the upper bound computation.

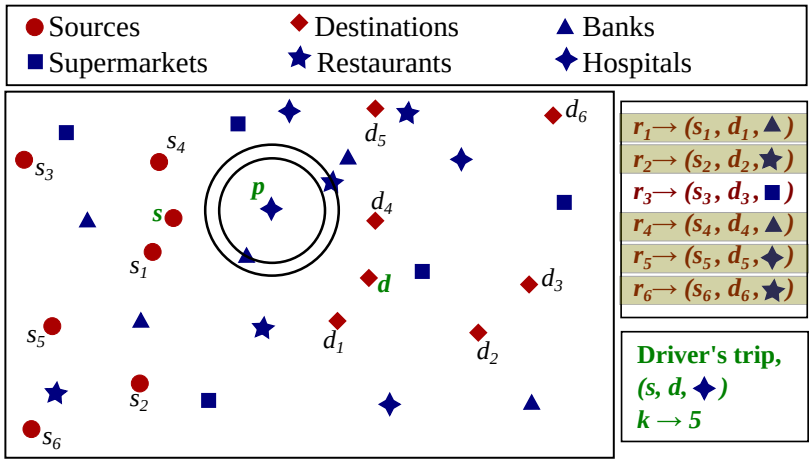
Then the technique computes the costs of the considered 5 riders for the ridesharing trip according to Equation 2.3. Let the costs be as follows: $C_1 = 40$, $C_2 = 75$, C_4



(a) The nearest POI, i.e., the driver's fixed hospital p is considered and the rider r_5 is selected



(b) The nearest POI, i.e., the nearest bank is retrieved and the riders r_1 and r_4 are selected



(c) The nearest POI, i.e., the nearest restaurant is retrieved and the riders r_2 and r_6 are selected

Figure 4.3: Upper bound computation for $k = 5$

= 45, C_5 = 75, C_6 = 100, where C_i represents the cost for rider r_i. Based on these costs, a ridesharing group $\Gamma^h = \{r_1, r_2, r_4, r_5\}$ of size $n = 4$ is formed with n lowest costs and the cost C^h of the ridesharing group Γ^h is $C^h = 40 + 45 + 75 + 75 = 235$. Thus the upper bound of the optimal ridesharing group cost, $h = C^h = 235$.

4.4 Lower Bound Computation

In this section, we present a heuristic technique to compute the lower bound L_i of the optimal group cost if a ridesharing group includes a specific rider r_i. To compute L_i, for the ridesharing group of size n, r_i's cost (either actual or the lower bound of the cost) for the ridesharing trip and the smallest costs (either actual or the lower bound of the cost) for other $n - 1$ members are considered.

To compute the upper bound of the optimal group cost, nearest POIs from the driver's fixed POI location p for at least $x\%$ of the required number of POI types have been already retrieved from the database. Hence if a rider r_i's required POI-type is retrieved in this process, then the rider's cost can be computed accurately using d_{t_i}, where d_{t_i} is the shortest distance between p and r_i's POI type. On the other hand, if a rider r_i's POI-type has not been retrieved during the upper bound computation, then the distance of such POI type must be greater than the distance of the last retrieved POI from the database. Let the distance can be denoted as d^{max}. Thus the lower bound of the corresponding rider's cost can be computed using d^{max} instead of using the real POI distance d_{t_i}. Then with these computed costs (either actual or the lower bound of the cost) of the riders, L_i can be computed. Specifically, for a group size n, L_i is the summation of the cost of r_i and $n - 1$ smallest costs of other riders.

Now, consider the same example scenario of Figure 4.2. In the example, the car capacity, $k = 5$, the ridesharing group size, $n = k - 1 = 4$, and the computed upper bound of the optimal group cost, $h = 235$. During the upper bound computation, 3 POI-types have been explored and thus the actual cost for all riders except r_3 can be computed (Figure 4.3). The costs for r_1, r_2, r_4, r_5, and r_6 are as follows: C_1 = 40, C_2 = 75, C_4 = 45, C_5 = 75, C_6 = 100. Let the distance d_{t_3} of the desired POI-type for r_3 be 25, and POIs up to the distance 15 meters has been retrieved, i.e.,

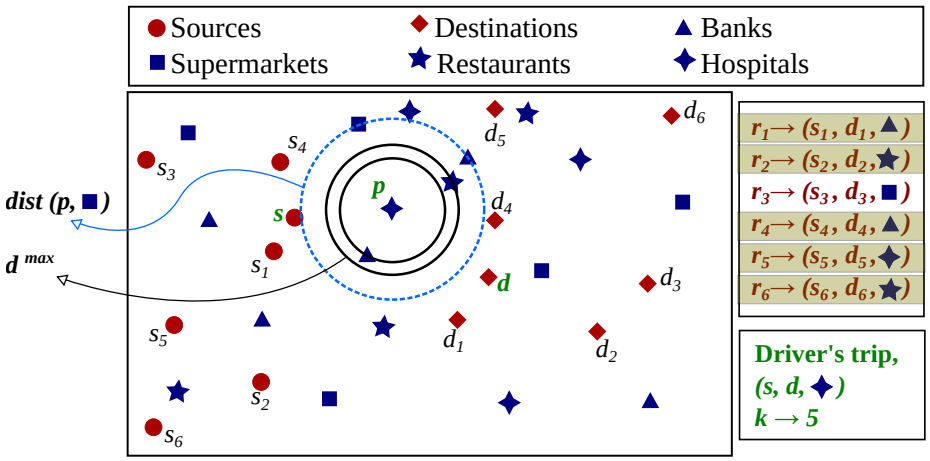


Figure 4.4: Lower bound computation of a POI-type's distance

$d^{max} = 15$ (Figure 4.4). Thus, the lower bound of the distance of the r_3's required POI-type is equal to d^{max}. If the distance χ_3 for the rider r_3 with respect to source and destination is $\chi_3 = dist(s, s_3) + dist(d, d_3) = 40$, then the lower bound of the rider r_3's cost is $C_3 = 40 + 2 \times d^{max} = 70$. Thus the derived lower bounds of the optimal group cost containing specific riders are as follows:

$$
\begin{aligned}
L_1 &= C_1 + 45 + 70 + 75 = 230 \\
L_2 &= C_2 + 40 + 45 + 70 = 230 \\
L_3 &= C_3 + 40 + 45 + 75 = 230 \\
L_4 &= C_4 + 40 + 70 + 75 = 230 \\
L_5 &= C_5 + 40 + 45 + 70 = 230 \\
L_6 &= C_6 + 40 + 45 + 70 = 255
\end{aligned}
$$

We observe that the lower bound of the optimal group cost containing the rider r_6 exceeds the computed upper bound h.

4.5 Pruning Riders

In ARGTP_EA, the riders are pruned in the following ways: (i) using the upper bound of the optimal group cost of the ridesharing group, and (ii) the lower bound of the optimal group cost of the ridesharing group that includes a specific rider, and (iii) using the pruned POI types. We discuss the first two techniques in the following subsections and the third technique in Section 4.6.

4.5.1 Using Upper Bound

We compute the maximum allowed distances γ_s, γ_d of the source and destination of a rider from the driver's source and destination, respectively, using the upper bound, and prune the riders based on γ_s, γ_d. If the ridesharing group size is n, these maximum allowed distances can be derived as follows:

- The maximum allowed distance of a rider's source from the driver's source γ_s is measured as the difference between the upper bound and the summation of the following values: (i) the smallest distances of $(n-1)$ riders' sources from the driver's source, (ii) the smallest distances of n riders' destinations from the driver's destination, and (iii) the smallest distances of n riders' POIs from the driver's fixed POI.

- The maximum allowed distance of a rider's destination from the driver's destination γ_d is measured as the difference between the upper bound and the summation of the following values: (i) the smallest distances of n riders' sources from the driver's source, (ii) the smallest distances of $(n-1)$ riders' destinations from the driver's destination, and (iii) the smallest distances of n riders' POIs from the driver's fixed POI.

We prune the riders whose source or destination is not located within the corresponding maximum allowable distance from the driver's source or destination, respectively.

To explain the pruning process, we consider the example scenario of Figure 4.2. Here, the upper bound of the optimal group cost, $h = 235$. Suppose we have

- The summation of smallest distances of $(n-1)$ riders' sources from the driver's source $s = 35$

- The summation of smallest distances of n riders' destinations from the driver's destination $d = 75$

- The summation of smallest distances of n riders' POIs from $p = 40$

Then the maximum allowed distance for a rider's source from the driver's source γ_s $= 235 - 35 - 75 - 40 = 85$.

Similarly we can compute γ_d. Suppose we have

- The summation of smallest distances of n riders' sources from the driver's source $s = 60$
- The summation of smallest distances of $(n-1)$ riders' destinations from the driver's destination $d = 55$
- The summation of smallest distances of n riders' POIs from $p = 40$

Then the maximum allowed distance for a rider's destination from the driver's destination $\gamma_d = 235 - 60 - 55 - 40 = 80$.

Thus, for the ARGTP query, we can prune a rider r_i if $dist\ (s,\ s_i) > 85$, or $dist\ (d_i,\ d) > 80$, or both of them.

The following lemma shows that if the source or the destination of a rider is located at a distance more than the maximum allowed distance from the driver's source or destination, respectively, then the rider cannot be a part of the optimal solution.

Lemma 4.5.1. *If the distance of a rider's source (or destination) from the driver's source (or destination) exceeds the maximum allowed distance, the rider cannot be a member of the optimal ridesharing group for an ARGTP query.*

Proof. Let there be a driver's trip from source s to destination d via a fixed POI p and a set of riders. The riders r_is are in the optimal ridesharing group Γ of size n and group cost of the optimal ridesharing group is C. Let there be a rider r_j $(r_j \notin \Gamma, i \neq j)$ with the source and destination pair $(s_j,\ d_j)$ in the riders-set whose $dist(s, s_j)$ (or $dist(d_j, d)$) exceeds the maximum allowed distance γ_s (or γ_d). C is measured as the summation of $\sum_{r_i \in \Gamma} C_i$. We have to show that $C_j > \max_{r_i \in \Gamma} C_i$.

C is measured as the summation of (i) $\sum_{r_i \in \Gamma} dist(s, s_i)$, (ii) $2 \times \sum_{r_i \in \Gamma} d_{t_i}$, and (iii) $\sum_{r_i \in \Gamma} dist(d_i, d)$. On the other hand, the maximum allowed distance γ_s is measured by deducting the following components from the upper bound of C: (i) the summation of the smallest possible distances that $(n-1)$ riders can have to travel to the driver's source from their sources, (ii) the summation of the smallest possible distances that n

riders can have to travel to and from the driver's fixed POI, and (iii) the summation of the smallest possible distances that n riders can have to travel from the driver's destination to their destinations. We know that $C_j = dist(s, s_j) + 2 \times d_{t_j} + dist(d_j, d)$. Thus, if $dist(s, s_j) > \gamma_s$, then $C_j > \max_{r_i \in \Gamma} C_i$ for d_{t_j} and $dist(d_j, d)$ having greater than or equal to the smallest one. Similarly we can show that if $dist(d, d_j) > \gamma_d$, then $C_j > \max_{r_i \in \Gamma} C_i$ for d_{t_j} and $dist(s_j, s)$ having greater than or equal to the smallest one. □

4.5.2 Using Lower Bound

A rider is pruned if the lower bound of the cost of a ridesharing group containing the rider exceeds the upper bound of the optimal group cost. In the example of Section 4.4, we can observe that the lower bound of group cost containing the rider r_6 exceeds the upper bound of the optimal group cost, $h = 235$. Thus, a ridesharing group that includes r_6 cannot be the optimal ridesharing group and r_6 can be pruned.

4.6 Pruning POI-types

In addition to pruning the riders, the efficiency of an ARGTP query processing depends on the number of required POI-types to be retrieved from the POI database. The POI types are pruned in the following ways: (i) using the pruned riders, and (ii) refining the POI search space. The pruned POI types using the refined search space can further prune the riders who want to visit those pruned POI types.

After pruning the riders using the upper bound and the lower bounds, we prune a POI-type if all the riders who want to visit the POI-type are pruned. This pruning technique reduces the number of POI-types significantly. However, we can further prune the POI-types by refining the POI search space. The search space is pruned using the upper bound of the optimal group cost (see Section 4.3).

Considering the upper bound and the maximum slugging distance of the candidate riders, the maximum allowed distance γ_p for a POI from the driver's fixed POI is derived.

If the ridesharing group size is n, γ_p is measured by deducting the following components from the upper bound of the optimal group cost: (i) the smallest distances of n riders' sources from the driver's source, (ii) the smallest distances of n riders' destinations from the driver's destination, and (iii) the smallest distances of $(n-1)$ riders' POIs from the driver's fixed POI. If the maximum slugging distance max_slug of the candidate riders is less than γ_p, max_slug is assigned to γ_p. We refine the POI search space using γ_p and retrieve only those POIs that have distances from the driver's fixed POI less than or equal to γ_p.

For example, consider again the example scenario of Figure 4.2. the upper bound of the optimal group cost, $h = 235$. Suppose we have

- The summation of smallest distances of n riders' sources from the driver's source $s = 60$
- The summation of smallest distances of n riders' destinations from the driver's destination $d = 75$
- The summation of smallest distances of $(n-1)$ riders' POIs from $p = 20$

Then the maximum distance γ_p for a POI from the driver's fixed POI $= \frac{(235-60-75)}{2} - 20$ $= 30$.

Thus we retrieve only those POIs for different required POI-types located within 30 meters from the driver's fixed POI. If the maximum slugging distance of the candidate riders is 25, then we retrieve the POIs for different required POI-types located within 25 meters from the driver's fixed POI.

4.7 Optimal Ridesharing Group Computation

After pruning the riders and POI types, ARGTP_EA computes cost for the remaining riders and sorts the set of riders in an ascending order based on the computed costs for the ridesharing trip. Now, for the car capacity k, the optimal ridesharing group is computed with the first $k-1$ riders from the sorted set of riders.

4.8 Algorithms for Efficient Approach

The efficiency of the ARGTP algorithms depends on the number of riders and the number of POI-types. Thus our efficient approach (ARGTP_EA) applies some novel pruning techniques to prune significant number of riders and POI-types. In this section, we present the algorithms to evaluate ARGTP queries efficiently. Our algorithm uses best first search (BFS) to incrementally retrieve nearest POIs from the data storage. We assume that, POIs are indexed using an R^*-tree in the database. Our algorithm retrieves POIs upto a derived maximum distance for a POI and determines the optimal ridesharing group for the ARGTP query. We have proved that no rider or POI-type is pruned by our algorithms that can be a part of the optimal ridesharing group.

Table 4.1 summarizes the symbols we used for the algorithms:

Table 4.1: Symbols and their meanings

Symbols	**Meanings**
X	A set of riders' cost with respect to source and destination, i.e., $X = \{\chi_i\}, \forall r_i \in R$
Δ_p	A set of (p_{t_i}, d_{t_i}, t_i) pairs, where, d_{t_i} is the distance of the nearest POI p_{t_i} of type t_i from the driver's fixed POI p
d^{max}	The distance of the last retrieved POI from the POI database
max_slug	The maximum slugging distance of the riders $r_i \in R$
γ_p	The maximum allowable distance for a POI from the driver's fixed POI p, where, $\gamma_p \leq max\_slug$
γ_s, γ_d	The maximum allowable distance for riders' source and destination from the driver's source s and destination d
$GrpC_{min}$	The minimum cost with respect to source and destination for a group of n riders
L	A set of lower bounds L_i for riders $r_i \in R$

The following functions are used by the algorithms described in this section:

Compute_Dist (s, d, R)**:** Computes $dist(s, s_i)$ and $dist(d, d_i)$ for each rider $r_i \in R$, and stores the values in Δ_s and Δ_d respectively.

Compute_h (CL, n)**:** Computes the upper bound h of the optimal group cost of size n using the lowest n riders' cost from the set CL.

Compute_MinCost (R_T, Δ_p, n)**:** Considers the minimum d_{t_i} values for t_i POI-type, where $(p_{t_i}, d_{t_i}, t_i) \in \Delta_p$, and the riders' set R_T to compute the minimum distance required to travel from p by n riders.

Compute_Riders_Cost (s, d, τ, Δ_p)**:** Computes the cost of riders $r_i \in \tau$ such that $C_i = dist(s, s_i) + dist(d, d_i) + 2 \times d_{t_i}$, where $(p_{t_i}, d_{t_i}, t_i) \in \Delta_p$.

FindMin (X, n)**:** Returns the summation of minimum n values from the set X.

RefineRiders (R, Δ_p)**:** Removes a rider r_i from the set of riders R, if r_i want to visit a POI-type t_i and there is no such POI in the set Δ_p, i.e., $(p_{t_i}, d_{t_i}, t_i) \notin \Delta_p$.

RefineTypes (R_T, T)**:** Removes a POI-type t_i from the set of POI-types T, if all the riders of R_{t_i} who want to visit the POI-type are pruned, i.e., $R_{t_i} \notin R_T$.

Retrieve_NextPOIs $(p, T, \Delta_p, d^{max}, \eta)$**:** Starting from d^{max}, incrementally retrieves the η number of nearest unique POIs p_{t_i} of types $t_i \in T, (p_{t_i}, d_{t_i}, t_i) \notin \Delta_p$ with respect to the POI p, and returns the retrieved POI-set P', the set $\mathbb{D}$ of corresponding distance and the distance d^{max} of the last retrieved POI.

Retrieve_POIs $(p, d^{max}, \gamma_p, T, \Delta_p)$**:** Starting from d^{max}, incrementally retrieves the nearest unique POIs p_{t_i} of types $t_i \in T, (p_{t_i}, d_{t_i}, t_i) \notin \Delta_p$ with respect to the POI p upto the maximum distance γ_p and updates Δ_p accordingly.

In Section 4.8.1, we briefly discuss the algorithm to process ARGTP queries efficiently by our ARGTP_EA approach as we mentioned in Section 4.1. Then we present the details of the function to compute the upper bound in Section 4.8.2, the function to compute the lower bound in Section 4.8.3, the function to prune riders in Section 4.8.4 and the function to prune POI-types in Section 4.8.5.

4.8.1 Algorithm ARGTP_EA

Algorithm 1 describes the process to determine the optimal ridesharing group Γ for an ARGTP query that minimizes the group cost function C. The inputs of the algorithm are s, d, p, k, st, ht, R, x, where $\langle s, d, p, k, st, ht\rangle$ is a driver's trip, R is the set of available riders' trips in the system and x is the factor for processing the ARGTP query. The algorithm returns the optimal ridesharing group Γ with the set of corresponding POIs $\mathbb{P}$ and the optimal group cost C.

Algorithm 1 ARGTP_EA

Input: $s, d, p, k, st, ht,\ R,\ x$
Output: $\Gamma, \mathbb{P}, C$
1: $n \leftarrow k-1, X \leftarrow \phi, \Lambda \leftarrow \phi, P' \leftarrow \phi$
2: $R \leftarrow TrivialPruning(s, d, st, ht, R)$
3: $Initialize(R, R_T, T)$
4: **for** each $r_i \in R$ **do**
5: $\quad \chi_i \leftarrow dist(s_i, s) + dist(d, d_i)$
6: $\quad X \leftarrow X \cup \{\chi_i\}$
7: **end for**
8: $\langle h, d^{max}, \Delta_p\rangle \leftarrow Compute\_Upper\_Bound(s, d, p, n, R_T, T, x)$
9: $L \leftarrow Compute\_Lower\_Bound(n, R, X, d^{max}, \Delta_p)$
10: $\langle R, R_T\rangle \leftarrow Prune\_Riders(s, d, n, R, R_T, h, L, \Delta_p)$
11: $\langle \gamma_p, T\rangle \leftarrow Prune\_POItypes(n, R, R_T, T, h, X, \Delta_p)$
12: $\Delta_p \leftarrow Retrieve\_POIs(p, d^{max}, \gamma_p, T, \Delta_p)$
13: $R \leftarrow RefineRiders(R, \Delta_p)$
14: **for** each $r_i \in R$ **do**
15: $\quad C_i \leftarrow \chi_i + 2 \times d_{t_i}$
16: $\quad \Lambda \leftarrow \Lambda \cup \{C_i\}$
17: $\quad P' \leftarrow P' \cup \{p_{t_i}\}$
18: **end for**
19: $\langle \Gamma, \mathbb{P}, C\rangle \leftarrow Compute\_Optimal\_Group(k, R, \Lambda, P')$
20: **return** $\Gamma, \mathbb{P}, C$

The algorithm starts with initializing the group size n to $k-1$ and the sets X, Λ, P' to ϕ, where, X is the set of riders' cost with respect to source and destination, i.e., $X = \{\chi_i\}, \forall r_i \in R$, Λ is the set of riders' cost, and P' is the set of retrieved POIs. After the initialization, using Function *TrivialPruning*, some riders are trivially pruned according to time, slugging distance and threshold distance constraints. Then

the set R_T of riders-set R_{t_i} (R_{t_i} is a set of riders who want to visit a POI-type t_i) and the set of unique POI-types T are initialized using Function *Initialize*. Then for each rider $r_i \in R$, χ_i is calculated and X is updated accordingly.

In the next step, the algorithm computes the upper bound h of optimal ridesharing group cost using Function *Compute_Upper_Bound*. The function returns the upper bound h, the set Δ_p of (p_{t_i}, d_{t_i}, t_i) for each nearest retrieved POI p_{t_i} of type t_i, and the distance d^{max} of the last retrieved POI. The details of the function is presented in Section 4.8.2. Then the algorithm computes the set L of lower bounds L_i of the group containing the rider r_i using Function *Compute_Lower_Bound*. Section 4.8.3 shows the algorithm of the function. After that, using the upper bound h and the lower bound L, Function *Prune_Riders* prunes riders from R and R_T who cannot be members of the optimal ridesharing group. Then using Function *Prune_POItypes*, the algorithm prunes some POI-types from T and also determines the maximum distance γ_p for a POI-type to be retrieved from the driver's fixed POI p. Thus the algorithm prunes the set of riders R and the set of POI-types T as described in Section 4.5 and Section 4.6. The functions *Prune_Riders* and *Prune_POItypes* are briefly described in Section 4.8.4 and Section 4.8.5 respectively.

With the refined sets of riders and POI-types, then the algorithm retrieves the required POIs to compute the optimal ridesharing group using Function *Retrieve_POIs* that incrementally retrieves POIs from the POI database upto the maximum distance γ_p and returns the set Δ_p such that within γ_p the nearest POI-types $t_i \in T$ from the fixed POI p are retrieved. After that, the algorithm refines the set of riders R whose POI-types are not retrieved using Function *RefineRiders*. Then in Lines 14–18 of the algorithm, for each rider $r_i \in R$, the riders' cost C_i for the ridesharing trip are calculated, and the sets Λ and P' are updated accordingly. Finally, using Function *Compute_Optimal_Group*, the algorithm sorts the riders' list based on the calculated cost C_i and determines the optimal ridesharing group Γ along with the corresponding POI-set $\mathbb{P}$ and the optimal group cost C.

4.8.2 Function Compute_Upper_Bound

The key idea behind the efficiency of our ARGTP_EA approach is to prune a significant number of riders and POI-types. The pruning techniques require an upper bound of the optimal ridesharing group cost, and so at first our ARGTP_EA approach computes the upper bound.

Algorithm 2 Compute_Upper_Bound

Input: s, d, p, n, R_T, T, x
Output: h, d^{max}, Δ_p

1: $\tau \leftarrow \phi, \Delta_p \leftarrow \phi, d^{max} \leftarrow 0$
2: $\eta \leftarrow |T| \times x\%$
3: **while** $|\tau| < n$ **do**
4: $\quad \langle p_{t_j}, d_{t_j}, d^{max} \rangle \leftarrow Retrieve\_NextPOIs(p, T, \Delta_p, d^{max}, 1)$
5: $\quad \Delta_p \leftarrow \Delta_p \cup \{(p_{t_j}, d_{t_j}, t_j)\}$
6: $\quad \tau \leftarrow \tau \cup R_{t_j}$
7: **end while**
8: **if** $|\Delta_p| < \eta$ **then**
9: $\quad \eta \leftarrow \eta - |\Delta_p|$
10: $\quad \langle P', \mathbb{D}, d^{max} \rangle \leftarrow Retrieve\_NextPOIs(p, T, \Delta_p, d^{max}, \eta)$
11: $\quad$ **for** each $p_{t_j} \in P', d_{t_j} \in \mathbb{D}$ **do**
12: $\quad\quad \Delta_p \leftarrow \Delta_p \cup \{(p_{t_j}, d_{t_j}, t_j)\}$
13: $\quad\quad \tau \leftarrow \tau \cup R_{t_j}$
14: $\quad$ **end for**
15: **end if**
16: $CL \leftarrow Compute\_Riders\_Cost(s, d, \tau, \Delta_p)$
17: $h \leftarrow Compute\_h(CL, n)$
18: **return** h, d^{max}, Δ_p

The details of the upper bound computation is discussed in Section 4.3. Algorithm 2 shows the pseudocode to compute the upper bound of the optimal ridesharing group cost. The function takes s, d, p, n, R_T, T, x as inputs, where s, d, p are the driver's source, destination and fixed POI respectively, n is the ridesharing group size, R_T is the set of riders-set R_{t_i}, T is the set of unique POI-types and x is the factor to process the ARGTP query. The algorithm returns the upper bound h of the optimal group cost, the set Δ_p of (p_{t_i}, d_{t_i}, t_i) for each nearest retrieved POI p_{t_i} of type t_i, and the distance d^{max} of the last retrieved POI.

At first, the algorithm initializes τ, Δ_p to ϕ, d^{max} to 0 and η to $|T| \times x\%$, where τ is a set of riders considered to compute h and η is the minimum number POI-types that we aim to retrieve during the upper bound computation. After the initialization, the function retrieves the nearest POI p_{t_j} of POI-type $t_j \in T$ from p using *Retrieve_NextPOIs* and updates d^{max}, Δ_p, τ accordingly. The function incrementally retrieves nearest POIs from p until a group of size n can be formed from the set τ of considered riders, i.e., $|\tau| < n$ (Lines 3–7). Then the function checks whether at least η number of POI-types have already been retrieved or not. If $|\Delta_p| < \eta$, then Function *Retrieve_NextPOIs* incrementally retrieves POI-types until total η number of POI-types are retrieved, and stores these nearest POIs in P' and the corresponding distances in $\mathbb{D}$. In the next step, the function updates the set Δ_p and τ according to P' and $\mathbb{D}$ (Lines 11–14). Then using Function *Compute_Riders_Cost*, the function computes the rider's cost for each rider $r_i \in \tau$ to join the driver's trip, and sorts the considered riders' list to store them in a sorted list CL. Finally, from the sorted list CL, Function *Compute_h* determines the upper bound h of the optimal group cost of size n.

4.8.3 Function Compute_Lower_Bound

The pruning techniques require the lower bound of group cost containing a rider (see Section 4.4). Algorithm 3 shows the pseudocode to compute the lower bound of group cost containing a rider. The inputs of the function are $n, R, X, d^{max}, \Delta_p$, where n is the ridesharing group size and R is a set of available riders and X is a set of riders' cost with respect to source and destination, i.e., $X = \{\chi_i\} = \{(dist(s, s_i) + dist(d, d_i))\}, \forall r_i \in R$. The function returns the a set L of lower bound L_i for each rider $r_i \in R$.

The algorithm starts with initializing a priority queue SL of (C'_i, r_i) pair with key C'_i for each rider r_i's computed cost C'_i. Then in Lines 2–9, for each rider, the function checks Δ_p whether the required POI-type is already retrieved or not, computes the rider's cost and includes the (C'_i, r_i) pair in SL. If $d_{t_i} \neq \infty$, then the POI-type is retrieved and d_{t_i} is used to compute the rider's cost. If $d_{t_i} == \infty$, then the POI-type is not retrieved yet, and so it requires at least the distance of d^{max}. So

Algorithm 3 Compute_Lower_Bound

Input: n, R, X, d^{max}, Δ_p
Output: L

1: $SL \leftarrow \phi$
2: **for** each $r_i \in R$ **do**
3: **if** $(p_{t_i}, d_{t_i}, t_i) \in \Delta_p$ **then**
4: $C'_i \leftarrow \chi_i + 2 \times d_{t_i}$
5: **else**
6: $C'_i \leftarrow \chi_i + 2 \times d^{max}$
7: **end if**
8: $SL \leftarrow SL \cup \{(C'_i, r_i))\}$
9: **end for**
10: **for** each $r_i \in R$ **do**
11: $count \leftarrow 0, j \leftarrow 0, L_i \leftarrow C'_i$
12: **while** $count < n - 1$ **do**
13: $(\upsilon, \tau) \leftarrow ElementAt(SL, j)$
14: **if** $r_i \neq \tau$ **then**
15: $L_i \leftarrow L_i + \upsilon$
16: $count \leftarrow count + 1$
17: **end if**
18: $j \leftarrow j + 1$
19: **end while**
20: $L \leftarrow L \cup \{L_i\}$
21: **end for**
22: **return** L

d^{max} is used instead of d_{t_i} to compute the lower bound of the rider's cost. After that, in Lines 10–21, the algorithm computes L_i for each rider $r_i \in R$ using the rider's cost C'_i and the priority queue SL as we discussed in Section 4.4.

4.8.4 Function Prune_Riders

Algorithm 4 Prune_Riders

Input: s, d, n, R, R_T, h, L, Δ_p
Output: R, R_T
1: $\langle \Delta_s, \Delta_d \rangle \leftarrow Compute\_Dist(s, d, R)$
2: $NC_{min} \leftarrow Compute\_MinCost(R_T, \Delta_p, n)$
3: $SD_{min} \leftarrow FindMin(\Delta_s, n-1) + FindMin(\Delta_d, n)$
4: $\gamma_s \leftarrow h - (SD_{min} + NC_{min})$
5: $SD_{min} \leftarrow FindMin(\Delta_s, n) + FindMin(\Delta_d, n-1)$
6: $\gamma_d \leftarrow h - (SD_{min} + NC_{min})$
7: **for** each $r_i \in R$, $L_i \in L$ **do**
8: **if** $dist(s_i, s) > \gamma_s$ **or** $dist(d, d_i) > \gamma_d$ **then**
9: $Remove(r_i, R, R_T)$
10: **else if** $L_i > h$ **then**
11: $Remove(r_i, R, R_T)$
12: **end if**
13: **end for**
14: **return** R, R_T

Algorithm 4 presents the pseudocode of the process to prune riders. It takes $s, d, n, R, R_T, h, L, \Delta_p$ as the inputs and returns the refined sets R, R_T after pruning riders. At first, the function computes $dist(s, s_i) \in \Delta_s$ and $dist(d, d_i) \in \Delta_d$ for each rider $r_i \in R$ using Function *Compute_Dist*. Then the function determines the minimum distance to travel from p for n riders using Function *Compute_MinCost*. After that, the function computes the maximum allowed distance γ_s and γ_d for a rider's source and destination from the driver's source s and destination d respectively. Then in Lines 7–13, the function prunes a rider r_i if any of the constraints, $dist(s, s_i) \leq \gamma_s$, $dist(d, d_i) \leq \gamma_d$ and $L_i \leq h$ violates. Finally, the function returns the refined riders' sets R, R_T.

4.8.5 Function Prune_POItypes

Algorithm 5 Prune_POItypes

Input: n, R, R_T, T, h, X, Δ_p
Output: γ_p, T

1: $max\_slug \leftarrow 0$
2: $T \leftarrow RefineTypes(R_T, T)$
3: $NC_{min} \leftarrow Compute\_MinCost(R_T, \Delta_p, n-1)$
4: $GrpC_{min} \leftarrow FindMin(X, n)$
5: $\gamma_p \leftarrow (h - GrpC_{min})/2 - NC_{min}$
6: **for** each $r_i \in R$ **do**
7: **if** $sd_i > max\_slug$ **then**
8: $max\_slug \leftarrow sd_i$
9: **end if**
10: **end for**
11: **if** $max\_slug < \gamma_p$ **then**
12: $\gamma_p \leftarrow max\_slug$
13: **end if**
14: **return** γ_p, T

Algorithm 5 shows the pseudocode of the process to prune POI-types. It takes $n, R, R_T, T, h, X, \Delta_p$ as the inputs, and returns the maximum allowed distance γ_p for a POI and the refined set of POI-types T. The algorithm starts with initializing the maximum slugging distance max_slug of the riders $r_i \in R$ to 0, and then refining the set T according to the refined set R_T. Then the function determines the minimum distance to travel from p for $n-1$ riders using Function *Compute_MinCost*, and also the minimum cost of a group of n riders with respect to source and destination using *FindMin* from X. After that, the function determines maximum distance γ_p (Line 5). In the next step, the function computes the maximum slugging distance max_slug of riders $r_i \in R$ (Lines 6–10). Finally, the derived maximum distance γ_p is compared with max_slug and if $max\_slug < \gamma_p$, then γ_p is replaced with max_slug.

4.9 Proof of Correctness

Theorem 4.9.1. *Prune_Riders prunes only those riders who cannot be members of the optimal ridesharing group of size n for an ARGTP query.*

Proof. If the optimal ridesharing group of size n is Γ, and the upper bound of the optimal group cost is h, then the optimal cost of Γ is $C \leq h$. Let there be a rider $r_j \in R, r_j \notin \Gamma$ who is pruned by Algorithm 4. According to the algorithm, the rider r_j can be pruned, if any of the followings is true: (i) $dist(s, s_j)$ exceeds the maximum allowed distance γ_s, (ii) $dist(d, d_j)$ exceeds the maximum allowed distance γ_d, (iii) the lower bound L_j exceeds the upper bound h. We have to show that $C_j > \max_{r_i \in \Gamma} C_i$.

Case (i), (ii): Let $dist(s, s_j) > \gamma_s$. We have proved in Lemma 4.5.1 that if a rider's source (or, destination) exceeds the maximum allowed distance γ_s (or, γ_d), she cannot be a member of the optimal ridesharing group. Thus $C_j > \max_{r_i \in \Gamma} C_i$, and the rider r_j can be pruned obviously. Similarly, if $dist(d, d_j) > \gamma_d$, then according to the lemma, $C_j > \max_{r_i \in \Gamma} C_i$ and so, the rider r_j can be pruned.

Case (iii): Let $L_j > h$, where h is the upper bound of the optimal group cost. The lower bound L_j is computed using the rider r_j's cost C_j, and the minimum cost of $(n-1)$ riders of the riders' set $R - \{r_j\}$. If the optimal ridesharing group cost is C, then $C \leq h$, whereas $L_j > h$. That's why, it is obvious that $C_j > \max_{r_i \in \Gamma} C_i$. □

Theorem 4.9.2. *The required POI-types to compute the optimal ridesharing group Γ of size n can be found within γ_p distance from the driver's fixed POI p.*

Proof. (By Contradiction) Let there be a POI p_{t_j} of type t_j such that $d_{t_j} > \gamma_p$, and the POI-type t_j is required by a rider $r_j \in R$. If the rider r_j can be in an optimal ridesharing group Γ' with the group cost C', then $C' \leq C$.

According to our assumption, $d_{t_j} > \gamma_p$. Let there be a rider $r^m \in R$ whose cost C^m is maximum in Γ, i.e., $C^m = \max_{r_i \in \Gamma} C_i$ and the distance of the POI-type required by r^m is $d_{t^m} \leq \gamma_p$. Let the rider r^m's cost with respect to source and destination locations be χ^m. The rider r_j can be in the optimal ridesharing group if $C_j \leq C^m$.

Since $d_{t_j} > \gamma_p$, it is obvious that $d_{t_j} > d_{t^m}$, and $\chi_j < \chi^m$. If $\Gamma' = \Gamma - \{r^m\} \cup \{r_j\}$, then the optimal cost C' will be

$$C' = \sum_{r_i \in \Gamma - \{r^m\}} C_i + \chi_j + 2 \times d_{t_j}.$$

If χ_j is in the lowest n values of X, then according to Algorithm 5,

$$GrpC_{min} + 2 \times NC_{min} \leq \sum_{r_i \in \Gamma - \{r^m\}} C_i + \chi_j,$$

and $d_{t_j} > \gamma_p$, thus

$$C' \geq GrpC_{min} + 2 \times NC_{min} + 2 \times d_{t_j} > h.$$

We know that $C \leq h$. Thus if $C' > h$, then $C' > C$, which contradicts our assumption. □

Theorem 4.9.3. *For an ARGTP query, if R is a set of n_r riders' trips, $\langle s, d, k, st, p, ht \rangle$ is a driver's trip and $U = \{U_1, U_2, \ldots, U_q\}$ is a set of possible ridesharing groups, where, $q = \binom{n_r}{k-1}$, then ARGTP_EA returns a ridesharing group $\Gamma \in U$ with the set of corresponding POIs $\mathbb{P}$ such that $C < C'$ for any group $\Gamma' \in U$ and $\Gamma \neq \Gamma'$.*

Proof. Let $\Gamma' \in U$ be a ridesharing group that is not returned by Algorithm 1. Let r_l be a rider that is not considered, where, $r_l \in \Gamma'$ and $r_l \notin \Gamma$. If r_l is considered, then Γ' has the minimum group cost C', i.e., $C' < C$. There can be two reasons that r_l is not considered: (i) r_l has been pruned for the current ARGTP query, and (ii) the required POI p_{t_l} of type t_l for the rider r_l is not retrieved. From Theorem 4.9.1 and Theorem 4.9.2, there can be no such rider r_l. Thus ARGTP_EA always returns a ridesharing group Γ with the corresponding POI-set $\mathbb{P}$ which minimizes the group cost function C.

□

4.10 Complexity Analysis

If there are $|P|$ number of POIs stored in R^*-tree and B is the branching factor, then the total number of pages on disk is $\kappa = |P|/B$. There are total $|R|$ number of available riders in the system and after applying the pruning techniques, there are n_r eligible riders and $|P'|$ number of POIs for an ARGTP query, where, $n_r \ll |R|$ and $|P'| \ll |P|$. Thus the maximum number of pages to be accessed is $\kappa' = |P'|/B$ and $\kappa' \ll \kappa$. Now, we can analyze the time complexity of the ARGTP_EA algorithm using the following lemmas.

Lemma 4.10.1. *The time complexity of Function Compute_Upper_Bound is* $O(\kappa)$.

Proof. The time complexity of Function Compute_Upper_Bound depends on the number of page access in the R^*-tree to retrieve $n', (1 \leq n' \leq n)$ or at least η number of POI-types. We use the incremental nearest neighbor search method to retrieve the POI-types. Thus the worst case time complexity to search these POI-types is $O(\kappa)$. □

Lemma 4.10.2. *The time complexity of Function Compute_Lower_Bound is* $O(|R| + |R| \log |R|)$.

Lemma 4.10.3. *The time complexity of Function Prune_Riders is* $O(|R| + |R| \log |R|)$.

Lemma 4.10.4. *The time complexity of Function Prune_POItypes is* $O(|R| + |R| \log |R|)$.

With the pruned set of n_r riders and the pruned set of $|P'|$ POIs, the time complexity of the ARGTP_EA algorithm is,

$$O(\kappa' + |R| + |R| \log |R|) \approx O(\kappa' + |R| \log |R|) \approx O(\kappa').$$

If we do not apply the pruning techniques, then the time complexity will be $O(\kappa) = O(|P|/B) \gg O(\kappa')$, which requires a lot of time to compute the ARGTP query.

Chapter 5

Baseline Approach

To the best of our knowledge, we develop the first approach ARGTP_EA to process ARGTP queries efficiently (see Chapter 4). Thus there exists no work to process ARGTP queries in the literature. To validate the efficiency of ARGTP_EA in experiments, we develop a baseline approach (ARGTP_BA) for processing ARGTP queries.

To process an ARGTP query, ARGTP_BA considers full set of available riders and required POI-types by the riders for computing the optimal ridesharing group, whereas, the efficient approach (ARGTP_EA) prunes a large number of riders and POI-types before computing the optimal ridesharing group. It is guaranteed that the pruned riders and POI-types cannot be part of the optimal solution. Figure 5.1 shows the steps of the ARGTP_BA approach.

In this approach, first of all, the nearest POIs for the required POI-types with respect to the driver's fixed POI location are retrieved from the database. Then the approach computes the cost of each available rider in the system for the ridesharing trip. After that, the riders' list is sorted according to the calculated costs of the riders in an ascending order. Finally, the ridesharing group is formed with the first $k-1$ riders from the riders' list and the ridesharing group is returned as the query answer.

However, the limitations of the ARGTP_BA approach is extremely high processing overhead for considering a large set of riders and their required POI-types to form the optimal ridesharing group. On the other hand, the pruning techniques used in the ARGTP_EA approach reduces the processing overhead significantly, and thus

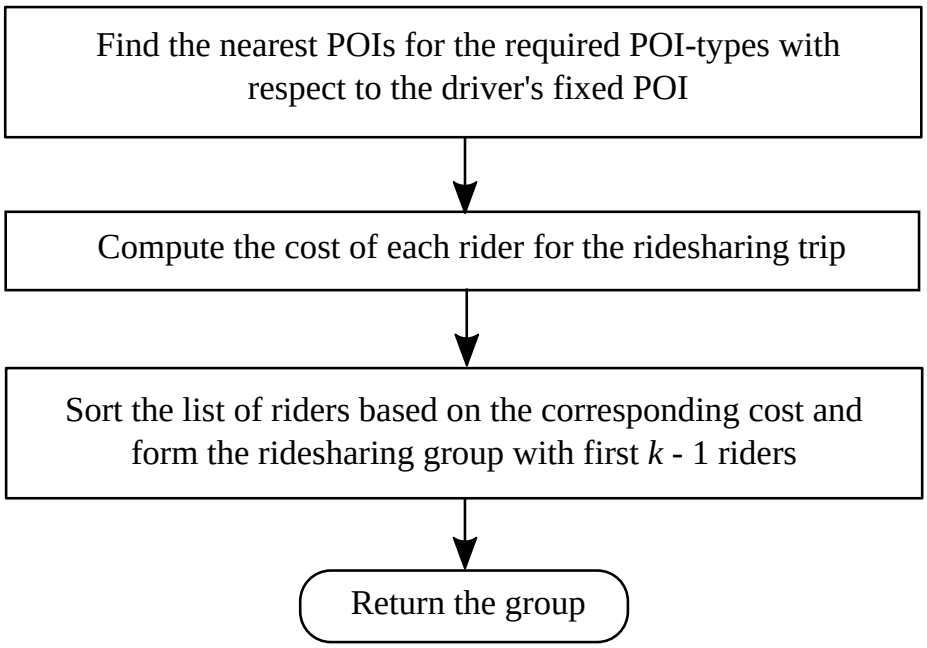


Figure 5.1: Overview of the ARGTP_BA approach

overcomes the limitations of the ARGTP_BA approach.

5.1 Algorithms for Baseline Approach

Algorithm 6 shows the process to determine the optimal ridesharing group Γ for an ARGTP query that minimizes the group cost function C. The inputs of the algorithm are s, d, p, k, st, ht, R, where $\langle s, d, p, k, st, ht \rangle$ is a driver's trip and R is the set of available riders' trips in the system. The algorithm returns the optimal ridesharing group Γ with the set of corresponding POIs $\mathbb{P}$ and the optimal ridesharing group cost C.

The algorithm starts with initializing the sets T, Δ_p, Λ, P' to ϕ and the values d^{max}, max_slug to 0, where T is a set of unique POI-types required by the riders of R, Δ_p is a set of (p_{t_i}, d_{t_i}, t_i) for the distance d_{t_i} of each nearest POI p_{t_i} of type t_i from the driver's fixed POI p, Λ is the set of riders' cost for the riders $r_i \in R$, P' is the set of POIs those are retrieved from the POI database, max_slug is the maximum

Algorithm 6 ARGTP_BA

Input: $s, d, p, k, st, ht,\ R$
Output: $\Gamma, \mathbb{P}, C$

1: $T \leftarrow \phi, \Delta_p \leftarrow \phi, \Lambda \leftarrow \phi, P' \leftarrow \phi$
2: $d^{max} \leftarrow 0, max\_slug \leftarrow 0$
3: $R \leftarrow TrivialPruning(s, d, st, ht, R)$
4: **for** each $r_i \in R$ **do**
5: **if** $sd_i > max\_slug$ **then**
6: $max\_slug \leftarrow sd_i$
7: **end if**
8: **if** $t_i \notin T$ **then**
9: $T \leftarrow T \cup \{t_i\}$
10: **end if**
11: **end for**
12: $\Delta_p \leftarrow Retrieve\_POIs(p, d^{max}, max\_slug, T, \Delta_p)$
13: **for** each $r_i \in R$ **do**
14: **if** $(p_{t_i}, d_{t_i}, t_i) \in \Delta_p$ **then**
15: $\chi_i \leftarrow dist(s_i, s) + dist(d, d_i)$
16: $C_i \leftarrow \chi_i + 2 \times d_{t_i}$
17: $P' \leftarrow P' \cup \{p_{t_i}\}$
18: **else**
19: $C_i \leftarrow \infty$
20: **end if**
21: $\Lambda \leftarrow \Lambda \cup \{C_i\}$
22: **end for**
23: $\langle \Gamma, \mathbb{P}, C \rangle \leftarrow Compute\_Optimal\_Group(k, R, \Lambda, P')$
24: **return** $\Gamma, \mathbb{P}, C$

slugging distance of riders $r_i \in R$, and $d^{max} = 0$ denotes that the distance of the last retrieved POI from the POI database is 0.

After the initialization, using Function *TrivialPruning*, some riders are trivially pruned according to the time, slugging distance and threshold distance constraints. Then the algorithm determines the maximum slugging distance max_slug of riders $r_i \in R$ and updates the POI-type set T accordingly. In the next step, the algorithm retrieves the required POIs to compute the optimal ridesharing group using Function *Retrieve_POIs* that incrementally retrieves POIs from the POI database upto the maximum slugging distance max_slug and returns the set Δ_p such that within max_slug the nearest POIs from p whose types are included in T are retrieved. Then in Lines 13–22 of the algorithm, for each rider $r_i \in R$, the rider's cost C_i for the ridesharing trip are calculated, and updates the sets Λ with the calculated cost C_i and P' if a POI p_{t_i} is retrieved from the POI database. Finally, using Function *Compute_Optimal_Group*, the algorithm sorts the riders' list based on the calculated cost C_i and determines the optimal ridesharing group Γ along with the set of corresponding POIs $\mathbb{P}$ and the optimal group cost C.

5.2 Complexity Analysis

If there are $|P|$ number of POIs stored in R^*-tree and B is the branching factor, then the total number of pages on disk is $\kappa = |P|/B$. Now, with a set of $|R|$ available riders in the system, the time complexity of ARGTP_BA algorithm is,

$$O(\kappa + |R| + |R| \log |R|) \approx O(\kappa + |R| \log |R|) \approx O(\kappa).$$

Thus the time complexity is $O(\kappa) = O(|P|/B)$, which requires a lot of time to compute the ARGTP query for a huge POI database and a large set of riders' trips.

Chapter 6

Experiments

In this chapter, we evaluate the performance of our efficient approach for processing ARGTP queries in experiments. Since there is no existing work for ARGTP queries, we compare our proposed efficient approach (ARGTP_EA) with the baseline approach (ARGTP_BA). The details of the approaches are described in Chapter 4 and Chapter 5 respectively.

In Section 6.1, we briefly describe the parameters of our experiment with their default values, and also the measures to evaluate the performance of our proposed approach. In the following Sections 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 present the effects of group size n, the number of riders in the system, the driver's trip length, the threshold distance $\lambda(\%)$, the factor $x(\%)$, the dataset size and the number of POI-types in the POI dataset respectively.

6.1 Experimental Setup

In this section, we present the details of the datasets (both POI and road network datasets) used for the experiments, different parameters varied in the experiments, and the measures for the comparative analysis.

6.1.1 Datasets

We use the California dataset [7] that consists of 63 types of 87635 POIs. The road network of California has 21048 nodes and 21693 edges, where a node represents a junction and an edge represents a road segment between two junctions in the road network. We generate the synthetic datasets of POIs of different types using the uniform random distribution. The data space is normalized to a span of 1000 $\times$ 1000 square units. An R^*-tree [61] is used to index all the POIs of a dataset and a in-memory graph data structure is used to store the road network.

We generate 100 sample drivers' trips (which initiate 100 ARGTP queries), and 8000 sample riders' trips, evaluate each experiments with 100 generated sample ARGTP queries and compute the average experimental results. For these trips, the source and destination locations are randomly generated over the road network, and the driver's fixed POI location and the rider's POI-type are selceted for the POI dataset using the uniform random distribution.

The algorithms described in this thesis are implemented in C++ platforms. We run all experiments using a computer with Intel Core i5 2.30 GHz CPU and 4GB RAM.

6.1.2 Parameters

In the experiments, we vary the following parameters: (i) the group size n, (ii) the number of riders in the system, (iii) the driver's trip length, (iv) the threshold distance λ (%), (v) the factor x(%), (vi) the dataset size, and (vii) the number of POI-types in a dataset. We have run each experiments using both California and synthetic datasets, and have recorded the average results as the performance measurements. Table 6.1 shows range and default values of each parameter. While varying a parameter, we set other parameters to their default values.

Table 6.1: Parameter settings

Parameter	**Range**	**Default**
Group Size, n	2, 3, 4, 5, 6, 7	4
Number of Riders	1k, 2k, 4k, 8k	2k
Driver's Trip Length	500, 1000, 1500, 2000, 2500	1500
Threshold Distance, $\lambda(\%)$	1, 2, 3, 4	2
$x(\%)$	10, 20, 30, 40, 50	30
Dataset Size	5k, 10k, 20k, 40k, 80k, 160k	20k
Number of POI-types	5, 10, 20, 40	10

6.1.3 Measures

We run extensive experiments to validate the efficiency of our approach and the effectiveness of ARGTP queries. We measure the the efficiency of our approach in terms of the query processing time and the I/O overhead, and estimate the effectiveness of ARGTP queries in terms of the quality loss (%) and the rider's drop rate (%).

6.1.3.1 Efficiency Measures

We measure the the efficiency of our approach in terms of the query processing time and the I/O overhead.

Processing Time: We record the execution time of an ARGTP query to be processed as the query processing time in seconds.

I/O Overhead: The I/O overhead is measured as the number of page access in the R^*-tree. We consider the page size as 1024 bytes in the R^*-tree.

6.1.3.2 Effectiveness Measures

We estimate the effectiveness of ARGTP queries in terms of the quality loss (%) and the rider's drop rate (%).

Quality Loss (%): We define the quality loss (%) as the percentage of the shortest trip distance that a rider needs to travel more for a ridesharing trip. Let a rider r_i want to visit a POI of type t_i on the way from her source s_i to destination d_i, and there be a driver's trip from source s to destination d via a fixed POI p. If the shortest trip length from s_i to d_i via the nearest POI of type t_i is $(\ell_{t_i})_{shortest}$ and the actual ridesharing trip length to get the driver's trip is $(\ell_{t_i})_{actual}$, then

$$\text{Quality loss (\%)} = \frac{(\ell_{t_i})_{actual} - (\ell_{t_i})_{shortest}}{(\ell_{t_i})_{shortest}} \times 100\%$$

where $(\ell_{t_i})_{actual}$ is measured as the summation of the following distances: (i) the distance between s and s_i, (ii) the distance between s and p, (iii) twice of the distance between p and the rider's POI, (iv) the distance between p and d, and (v) the distance between d and d_i.

Rider's Drop Rate (%): We measure the rider's drop rate (%) as the percentage of riders' trips that would fail to get a ridesharing trip if the flexibility in the POI selection is not allowed, i.e., if the nearest POI is not located within the specified slugging distance from the driver's fixed POI, then the rider's trip fails. For the ridesharing group of size n, if the number of riders who fail to get a ridesharing trip is n_f, then

$$\text{Rider's drop rate (\%)} = \frac{n - n_f}{n} \times 100\%$$

6.2 Effect of Group Size n

In this section, we study the effects of group size n at the time of processing ARGTP queries. We vary the group size n by 2, 3, 4, 5, 6, 7 using both California and synthetic datasets, and measure the processing time, the I/O overhead, the quality loss (%) and the rider's drop rate (%).

Figure 6.1 shows the performance for varying the group size n using California dataset (Figure 6.1(a)–(d)) and synthetic dataset (Figure 6.1(e)–(h)). For both datasets, we can observe that the processing time and I/O overhead slightly increase

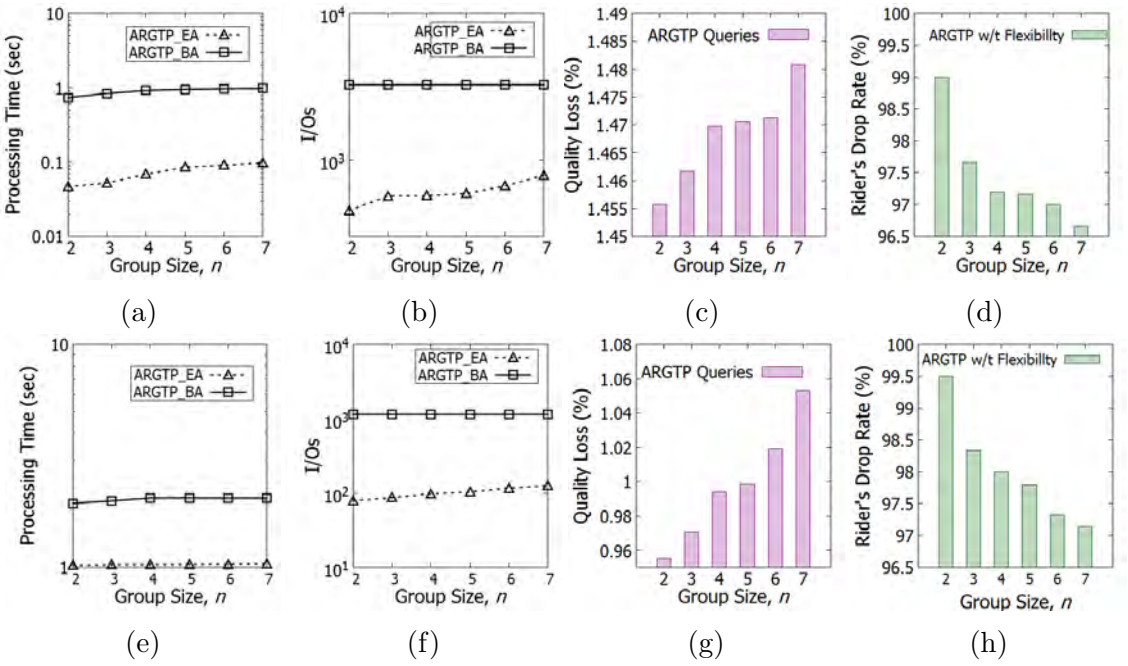


Figure 6.1: Effect of group size n using California (a–d) and synthetic (e–h) datasets

with the increase of group size n. The ARGTP_EA approach requires significantly less processing time and I/O overhead than the ARGTP_BA approach, as expected. The ARGTP_BA approach considers all available riders and their required POI-types to compute the optimal ridesharing group, whereas, the ARGTP_EA approach prunes significant number of riders and POI-types and thus determines the optimal ridesharing group efficiently.

On the other hand, quality loss (%) increases and the rider's drop rate (%) decreases slightly with the increase of group size n. With the increase of group size n, the optimal rideharing group cost increases which increases the average quality loss (%) in a ridesharing group. For example, the quality loss is 1.47% for $n = 6$, whereas the quality loss is 1.48% for $n = 7$. We can observe from the graphs that with a very low quality loss (%) (ranges from 0.95%–1.48%) and flexibility in the selection of POIs, ARGTP queries can ensure complete ridesharing trips. If the flexibility is not allowed, then about 97%–99% riders' trips can be dropped.

6.3 Effect of Number of Riders

In this section, we study the effect of number of riders available in the system at the time of processing ARGTP queries. We vary the number of riders by 1000, 2000, 4000, 8000 using both California and synthetic datasets, and measure the processing time, the I/O overhead, the quality loss (%) and the rider's drop rate (%).

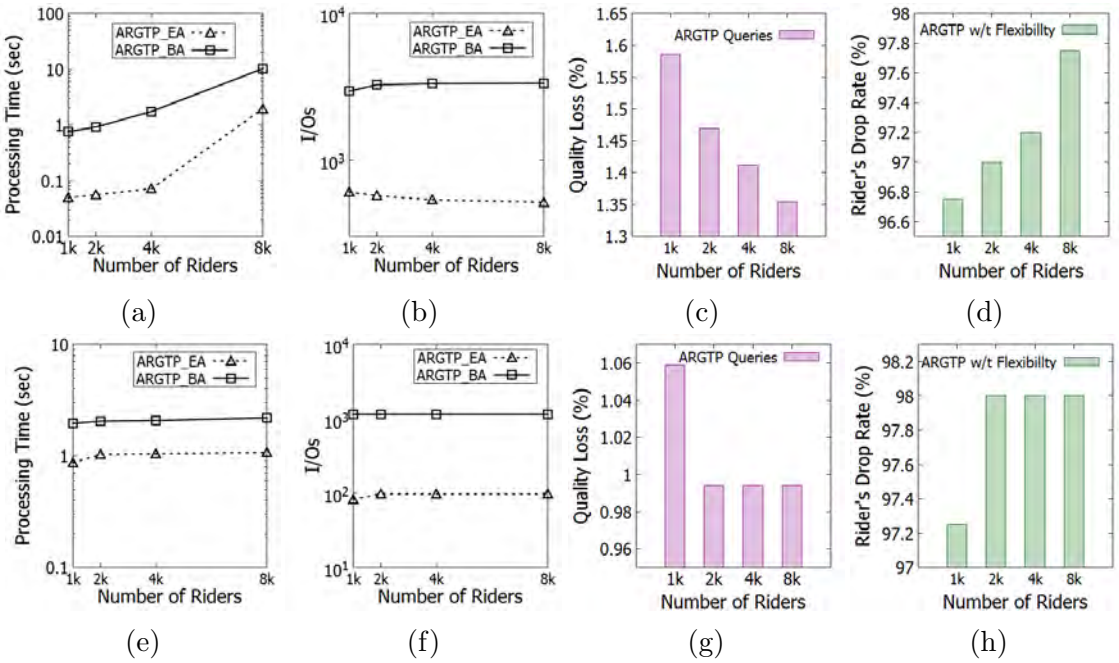


Figure 6.2: Effect of number of riders using California (a–d) and synthetic (e–h) datasets

Figure 6.2 shows the performance for varying the number of riders using California dataset (Figure 6.2(a)–(d)) and synthetic dataset (Figure 6.2(e)–(h)). We can observe that the performance is almost constant for synthetic dataset. Using California dataset, with the increase of the number of riders, the processing time increases and the I/O overhead for ARGTP_EA decreases, whereas the I/O overhead for ARGTP_BA remains almost constant. The reason behind such effect is that during the *upper bound computation*, the ARGTP_EA approach retrieves at least $x\%$ of POIs which is more effective for the large number of riders. That's why, the I/O overhead decreases for the ARGTP_EA approach. On ther other hand, to process larger

number of riders, the query requires more time. We can observe that the ARGTP_EA approach outperforms the ARGTP_BA approach in terms of the processing time and the I/O overhead.

Similarly for California dataset, quality loss (%) decreases and the rider's drop rate (%) increases slightly with the increase of the number of riders, as the candidate number of riders to compute the optimal ridesharing group cost increases. For example, the quality loss (%) decreases from 1.59% to 1.46% if the number of riders in increased from 1000 to 2000. We can observe from the graphs that using synthetic dataset, both of the quality loss (%) and the rider's drop rate (%) are almost constant except for 1000 number of riders in the system.

6.4 Effect of Driver's Trip Length

In this section, we study the impact of driver's trip length on the processing of ARGTP queries. The driver's trip length is the road network distance from the driver's source to the driver's destination via a fixed POI. All the group members share their trips for at least this length. We vary the driver's trip length by 500, 1000, 1500, 2000, 2500 using both California and synthetic datasets, and measure the processing time, the I/O overhead, the quality loss (%) and the rider's drop rate (%).

Figure 6.3 shows the performance for varying the driver's trip length using California dataset (Figure 6.3(a)–(d)) and synthetic dataset (Figure 6.3(e)–(h)). For both datasets, the processing time and I/O overhead decreases with the increase of the driver's trip length, as expected. The number of candidate riders decreases with the increase of the driver's trip length, which implies that the number of required POI-types also decreases and so do the I/O overhead and the processing time. On the other hand, the increasing trip length obviously increases the quality loss (%) and the rider's drop rate (%). Quality loss (%) is directly related to the driver's trip length, and so it increases with the increase of the driver's trip length.

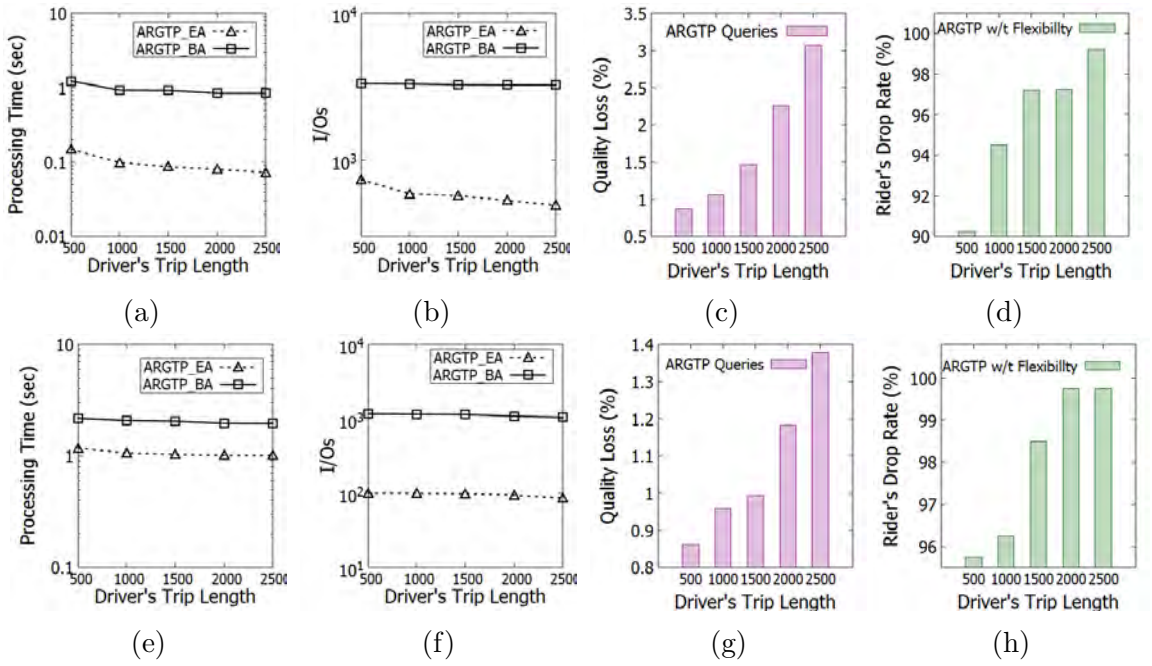


Figure 6.3: Effect of driver's trip length using California (a–d) and synthetic (e–h) datasets

6.5 Effect of Threshold Distance $\lambda(\%)$

In this section, we study the impact of threshold distance $\lambda(\%)$ on the processing of ARGTP queries. The threshold distance is specified by the riders. By specifying the threshold distance λ, a rider mentions that she can travel at most $\lambda\%$ more than the shortest trip to get a ridesharing trip. We vary the threshold distance by 1, 2, 3, 4 using both California and synthetic datasets. Thus in these experiments, we can consider only those riders whose threshold distance are within these values to process ARGTP queries.

Figure 6.4 shows the performance for varying the threshold distance using California dataset (Figure 6.4(a)–(d)) and synthetic dataset (Figure 6.4(e)–(h)). For both datasets, the processing time, I/O overhead, quality loss (%) and the rider's drop rate (%) increases with the increase of threshold distance. The reason behind such performance is that the increase of threshold distance increases the number of candidate riders for an ARGTP query, and so does the processing time. If the number of candidate rider increases, then the number of required POI-types also increases and

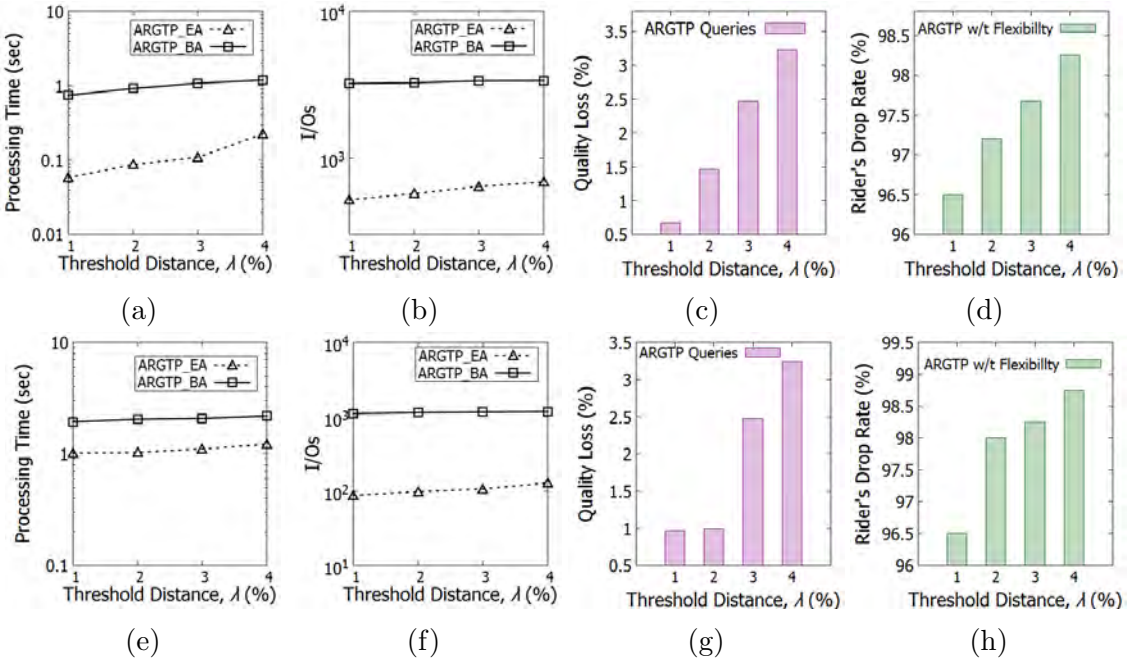


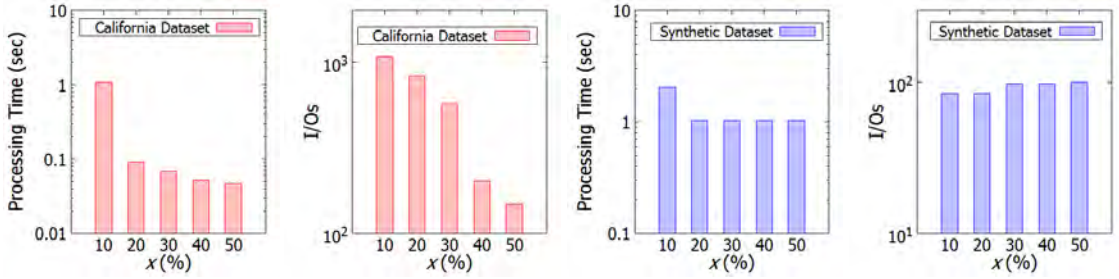
Figure 6.4: Effect of threshold distance $\lambda(\%)$ using California (a–d) and synthetic (e–h) datasets

so it requires more I/O access. On the other hand, if the rider's threshold distance increases, then it incurs more quality loss (%) (though it is very low, ranges from 96.5% to 98.5%).

6.6 Effect of $x(\%)$

In this section, we present the effect of $x(\%)$ at the time of processing ARGTP queries for the ARGTP_EA approach. To process an ARGTP query efficiently, the ARGTP_EA appproach retrieves at least $x\%$ of the required POI-types during the *upper bound computation*. We record the performance for both datasets in terms of processing time and I/O overhead.

Figure 6.5 shows the performance for varying $x\%$ using California dataset (Figure 6.5(a)–(b)) and synthetic dataset (Figure 6.5(c)–(d)). From the graphs, we can observe that both of the processing time and the I/O overhead decrease with the increase of $x\%$, except the I/O overhead for the synthetic dataset which is almost



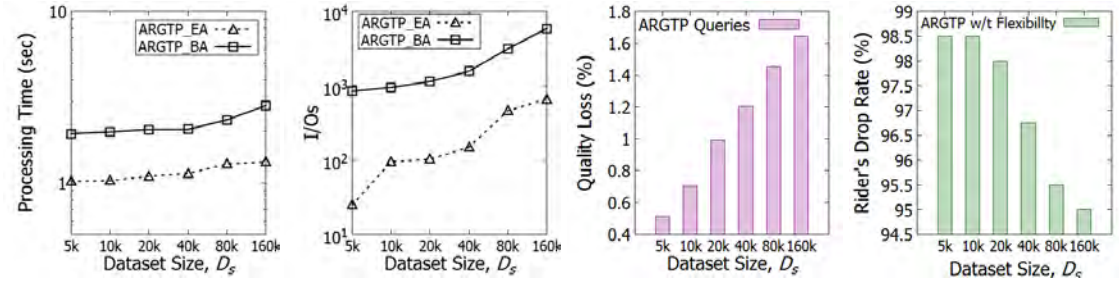
(a) (b) (c) (d)

Figure 6.5: Effect of $x(\%)$ using California (a–b) and synthetic (c–d) datasets

constant. The number of POI-types (5–40 POI-types) in synthetic dataset is much less than that (63 POI-types) of California dataset, and so $x\%$ does not have much significant effect for synthetic dataset.

6.7 Effect of Dataset Size

In this section, we study the impact of dataset size on the processing of ARGTP queries. To process an ARGTP query efficiently, we retrieve the required POI-types from the POI database. Our ARGTP_EA approach prunes the set of POI-types and thus refines the POI search space significantly. In these experiments, we observe the processing time and the I/O overhead to measure the efficiency of the ARGTP_EA approach.



(a) (b) (c) (d)

Figure 6.6: Effect of dataset size using synthetic (a–d) datasets

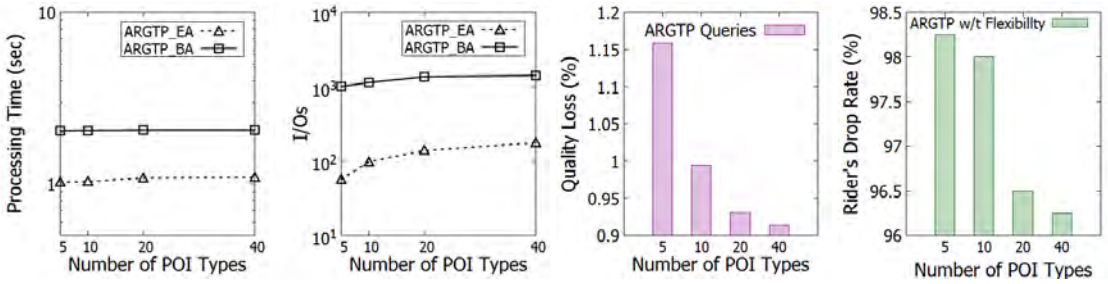
Figure 6.6 shows the performance for varying the dataset size using synthetic

dataset (Figure 6.6(a)–(d)). The processing time and the I/O overhead increase with the increase of dataset size, which is obvious. With the increase of dataset size, the POI search space increases which requires more I/O access and more processing time. We can observe that the ARGTP_EA approach outperforms the ARGTP_BA approach in terms of processing time and I/O overhead.

Similarly, the more the datasize is, the more options for a POI-type to satisfy is, which increases the rider's trip length and the quality loss (%) (only 0.5%–1.5% for varying dataset size 5k–40k). On the other hand, the increase of the dataset size decreases the rider's drop rate (%) at a low rate (ranges from 98.5% to 95%). With the increase of the dataset size, the probability of a POI for the required POI-type to be within the slugging distance increases, and so the rider's drop rate (%) decreases.

6.8 Effect of Number of POI-types

In this section, we present the effect of number of POI-types at the time of processing ARGTP queries. We vary the number of POI-types by 5, 10, 20, 40 using synthetic datasets, and measure the processing time, the I/O overhead, the quality loss (%) and the rider's drop rate (%).



(a) (b) (c) (d)

Figure 6.7: Effect of number of POI-types using synthetic (a–d) datasets

Figure 6.7 shows the performance for varying the number of POI-types using synthetic dataset (Figure 6.7(a)–(d)). The performance is similar as the performance for varying the dataset size, except the quality loss (%). With the increase of the number

of POI-types, the probability to get closer POIs for different POI-types required by the riders increases, which decreases the quality loss (%).

Chapter 7

Conclusion

In this thesis, we have introduced a new type of ridesharing query, an *Activity-aware Ridesharing Group Trip Planning (ARGTP)* query for road networks that enables a group of riders to get a complete ridesharing trip for both fixed and flexible locations and returns the optimal ridesharing group that minimizes the group cost function. If a rider want to visit a POI (e.g., any branch of a bank) in between her source location and destination location, then traditional ridesharing services require two separate ridesharing trips and there is no guarantee that the rider will get ridesharing options for both trips. Thus, to provide a rider a complete ridesharing trip via an intermediary POI, we introduce ARGTP queries. To process an ARGTP query, the spatial proximity of riders' trips are considered to match with that of the driver. In addition, ARGTP query allow riders to be flexible for visiting POI-types within their distance limits. For example, a rider may be happy to visit the second or third nearest branch of a bank instead of the first one, if in turn the probability to get a ridesharing trip is increased.

The major challenges to process ARGTP queries are to identify the optimal ridesharing group from a large set of riders and to explore the huge POI database for the required POI-types of the riders in the system. It requires high processing overhead and time to consider the large riders-set and the whole POI database for processing an ARGTP query. Thus the efficiency of processing an ARGTP query depends on the number of riders and the POI search space considered for processing an ARGTP query. We have proposed the first solution to process ARGTP queries efficiently. The key idea behind the efficiency is to prune significant number of riders

and POI-types that cannot be part of the ARGTP answer. We measured the efficiency of our approach in terms of processing time and I/O overhead to process an ARGTP query, and estimated the effectiveness of an ARGTP query in terms of the quality loss (%) and rider's drop rate (%).

Experiments show that the efficient approach requires on average 9 times less processing time and 11 times less I/O access than the baseline approach. In all experiments, we observed that the flexibility in selecting POIs incurs a very low quality loss (%) (ranges from 0.6% to 3.0%), and in return almost eliminates the rider's drop rate, whereas without allowing flexibility in selecting POIs causes 90%–99.5% rider requests to be dropped.

7.1 Future Challenges

In this research work, we have introduced an ARGTP query which enables a rider to visit an intermediary POI and allows flexibility in selecting the POI. However, ARGTP queries can be extended to visit more than one intermediary POIs. We have considered the *slugging* model for processing ARGTP queries, and there are other ridesharing models that can be applicable to process ARGTP queries. Thus some remarkable future challenges for this research are proposed below:

- In future, we aim to develop the solution to process ARGTP queries with multiple intermediary POI-types instead of a single one.
- In this thesis, we have considered a ridesharing model, slugging, where there is no detour required by the drivers. In future, we plan to apply other ridesharing models for processing ARGTP queries.
- We have involved a ridesharing service provider (RSP) in the system. Both riders and drivers submit their trip information to the RSP, and thus there is a possibility to reveal some private information from the RSP to others. To address these issues, we aim to develop a framework to process privacy-preserving ARGTP queries.

References

[1] S. Ma and O. Wolfson, "Analysis and evaluation of the slugging form of ridesharing," in *SIGSPATIAL/GIS*, 2013, pp. 64–73.

[2] E. Badger, "Slugging – the people's transit," March 2011.

[3] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte, "Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem," *Parallel Computing*, vol. 30, no. 3, pp. 377 – 387, 2004.

[4] K. Tsubouchi, K. Hiekata, and H. Yamato, "Scheduling algorithm for on-demand bus system," *Information Technology: New Generations*, vol. 10, pp. 189 – 194, May 2009.

[5] S. Yan and C.-Y. Chen, "An optimization model and a solution algorithm for the many-to-many car pooling problem," *Ann. Oper. Res.*, vol. 191, pp. 37 – 71, August 2011.

[6] Y. Wang, R. Kutadinata, and S. Winter, "Activity-based ridesharing: Increasing flexibility by time geography," in *GIS*, 2016, pp. 1:1–1:10.

[7] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *SSTD*, 2005, pp. 273–290.

[8] F. Spielberg and P. Shapiro, "Mating habits of slugs: Dynamic carpool formation in the i-95/i-395 corridor of northern virginia," *Transportation Research Record*, vol. 1711, pp. 31 – 38, 2000.

[9] (2010, June) Map of slugging sites in washington d.c. *slug-lines.com*. Forel Publishing Company, LLC. [Online]. Available: www.slug-lines.com

[10] E. Ahmadi and M. A. Nascimento, "A mixed breadth-depth first search strategy for sequenced group trip planning queries." in *MDM (1)*, 2015, pp. 24–33.

[11] T. Hashem, S. Barua, M. E. Ali, L. Kulik, and E. Tanin, "Efficient computation of trips with friends and families," in *CIKM*, 2015, pp. 931–940.

[12] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik, "Group trip planning queries in spatial databases," in *SSTD*, 2013, pp. 259–276.

[13] S. Samrose, T. Hashem, S. Barua, M. E. Ali, M. H. Uddin, and M. I. Mahmud, "Dynamic group trip planning queries in spatial databases," in *MDM*, 2015, pp. 122–127.

[14] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.

[15] D.-N. Yang, C.-Y. Shen, W.-C. Lee, and M.-S. Chen, "On socio-spatial group query for location-based social networks," in *KDD*, 2012, pp. 949–957.

[16] A. Tabassum, S. Barua, T. Hashem, and T. Chowdhury, "Dynamic group trip planning queries in spatial databases," in *SSDBM*, 2017, pp. 38:1–38:6.

[17] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *VLDB*, vol. 7, no. 14, pp. 2017–2028, 2014.

[18] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler, "Highly scalable trip grouping for large-scale collective transportation systems," in *EDBT*, 2008, pp. 678–689.

[19] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: Studies of ridesharing," in *IJCAI*, 2009, pp. 187–194.

[20] P. M. d'Orey, R. Fernandes, and M. Ferreira, "Empirical evaluation of a dynamic and distributed taxi-sharing system," in *ITSC*, 2012, pp. 140–146.

[21] A. P. Amey, "A proposed methodology for estimating rideshare viability within an organization, applied to the mit community," in *TRB Annual Meeting*, 2010, pp. 1–16.

[22] A. Kleiner, B. Nebel, and V. A. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," in *IJCAI*, 2011, pp. 266–272.

[23] M. Furuhata, M. Dessouky, F. Ordez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research*, vol. 57, pp. 28 – 46, 2013.

[24] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *ICDE*, 2013, pp. 410–421.

[25] T. Pedersen, "Cab-sharing: An effective, door-to-door, on-demand transportation service," in *ERTICO*, 03 2018, pp. 1–8.

[26] W. E. de Paepe, J. K. Lenstra, J. Sgall, R. A. Sitters, and L. Stougie, "Computer-aided complexity classification of dial-a-ride problems," *INFORMS Journal on Computing*, vol. 16, no. 2, pp. 120–132, 2004.

[27] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong, "An efficient ride-sharing framework for maximizing shared route," *TKDE*, vol. PP, pp. 1–1, 10 2017.

[28] W. Zhao, Y. Qin, D. Yang, L. Zhang, and W. Zhu, "Social group architecture based distributed ride-sharing service in vanet," *International Journal of Distributed Sensor Networks*, vol. 2014, pp. 1–8, 2014.

[29] M. Rigby, A. Krüger, and S. Winter, "An opportunistic client user interface to support centralized ride share planning," in *SIGSPATIAL*, 2013, pp. 34–43.

[30] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *European Journal of Operational Research*, vol. 223, no. 2, pp. 295 – 303, 2012.

[31] D. O. Santos and E. C. Xavier, "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem," in *IJCAI*, 2013, pp. 2885–2891.

[32] R. Baldacci, V. Maniezzo, and A. Mingozzi, "An exact method for the car pooling problem based on lagrangean column generation," *Operations Research*, vol. 52, no. 3, pp. 422–439, 2004.

[33] R. W. Calvo, F. de Luigi, P. Haastrup, and V. Maniezzo, "A distributed geographic information system for the daily car pooling problem," *Computers & Operations Research*, vol. 31, no. 13, pp. 2263 – 2278, 2004.

[34] N. Jing Yuan, Y. Zheng, L. Zhang, and X. Xie, "T-finder: A recommender system for finding passengers and vacant taxis," *TKDE*, vol. 25, no. 10, pp. 2390–2403, 2012.

[35] N. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang, "Sustainable passenger transportation: Dynamic ridesharing," Erasmus Research Institute of Management, Tech. Rep., 2009.

[36] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem (darp): Variants, modeling issues and algorithms," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 89–101, 2003.

[37] J. F. Cordeau and G. Laporte, "The dial-a-ride problem: Models and algorithms," *Annals of Operation Research*, vol. 153, no. 1, pp. 29 – 46, 2007.

[38] K. I. Wong and M. G. H. Bell, "Solution of the dial-a-ride problem with multi-dimensional capacity constraints," *International Transactions in Operational Research*, vol. 13, no. 3, pp. 195–208, 2006.

[39] Z. Xiang, C. Chu, and H. Chen, "A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints," *European Journal of Operational Research*, vol. 174, no. 2, pp. 1117 – 1139, 2006.

[40] J. W. B. JR., G. K. R. Kakivaya, and J. R. Sone, "Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing," *Engineering Optimization*, vol. 30, no. 2, pp. 91–123, 1998.

[41] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269 – 287, 1983.

[42] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Operations Research*, vol. 54, no. 3, pp. 573–586, 2006.

[43] P. Goel, L. Kulik, and K. Ramamohanarao, "Privacy-aware dynamic ride sharing," *TSAS*, vol. 2, no. 1, pp. 4:1–4:41, 2016.

[44] Y. Li, R. Chen, L. Chen, and J. Xu, "Towards social-aware ridesharing group query services," *TSC*, vol. PP, no. 99, pp. 1–1, 2017.

[45] Y. Wu, L. J. Guan, and S. Winter, "Peer-to-peer shared ride systems," in *GSN*, 2006, pp. 252–270.

[46] R. F. Teal, "Carpooling: Who, how and why," *Transportation Research Part A: General*, vol. 21, no. 3, pp. 203 – 214, 1987.

[47] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 1068–1080, 2008.

[48] R. K. Balan, K. X. Nguyen, and L. Jiang, "Real-time trip information service for a large taxi fleet," in *MobiSys*, 2011, pp. 99–112.

[49] K. Yamamoto, K. Uesugi, and T. Watanabe, "Adaptive routing of cruising taxis by mutual exchange of pathways," in *Knowledge-Based Intelligent Information and Engineering Systems*, I. Lovrek, R. J. Howlett, and L. C. Jain, Eds., 2008, pp. 559–566.

[50] D. Santani, R. K. Balan, and C. J. Woodard, "Spatio-temporal efficiency in a taxi dispatch system," 2008.

[51] D. Zhang, T. He, Y. Liu, and J. A. Stankovic, "Callcab: A unified recommendation system for carpooling and regular taxicab services," in *Int. Conf. on Big Data*, 2013, pp. 439–447.

[52] S. Ma and Y. Zheng, "Real-time city-scale taxi ridesharing," *TKDE*, vol. 27, no. 7, pp. 1782–1795, 2015.

[53] B. Jin and J. Hu, "Towards scalable processing for a large-scale ride sharing service," in *UIC/ATC*, 2012, pp. 940–944.

[54] Y. Dumas, J. Desrosiers, and F. Soumis, "The pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 54, no. 1, pp. 7 – 22, 1991.

[55] B. Cici, A. Markopoulou, E. Frias-Martinez, and N. Laoutaris, "Assessing the potential of ride-sharing using mobile and social data: A tale of four cities," in *UbiComp*, 2014, pp. 201–211.

[56] F. Bistaffa, A. Farinelli, and S. D. Ramchurn, "Sharing rides with friends: A coalition formation algorithm for ridesharing," in *AAAI*, 2015, pp. 608–614.

[57] S. Anwar, S. Nabila, and T. Hashem, "A novel approach for efficient computation of community aware ridesharing groups," in *CIKM*, 2017, pp. 1971–1974.

[58] K. Radke, M. Brereton, S. Mirisaee, S. Ghelawat, C. Boyd, and J. G. Nieto, "Tensions in developing a secure collective information practice - the case of agile ridesharing," in *INTERACT*, 2011, pp. 524–532.

[59] P. Cheng, H. Xin, and L. Chen, "Utility-aware ridesharing on road networks," in *SIGMOD*, 2017, pp. 1197–1210.

[60] A. K. M. M. R. Khan, O. Correa, E. Tanin, L. Kulik, and K. Ramamohanarao, "Ride-sharing is about agreeing on a destination," in *SIGSPATIAL/GIS*, 2017, pp. 6:1–6:10.

[61] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *SIGMOD*, 1990, pp. 322–331.