Cohesive Ridesharing Group Queries in Geo-Social Networks

CHANGBEOM SHIM1, GYUHYEON SIM2, AND YON DOHN CHUNG2, (Member, IEEE)

1Department of Electrical and Computer Engineering, Curtin University, Perth, WA 6102, Australia
2Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea

Corresponding author: Yon Dohn Chung (ydchung@korea.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grant NRF-2017R1A2A2A05069318.

ABSTRACT Ridesharing has gained much attention as a solution for mitigating societal, environmental, and economic problems. For example, commuters can reduce traffic jams by sharing their rides with others. Notwithstanding many advantages, the proliferation of ridesharing also brings some crucial issues. One of them is to rideshare with strangers. It makes someone feel uncomfortable or untrustworthy. Another complication is the high-latency of ridesharing group search because users may want to receive the result of their requests in a short time. Despite continuous efforts of academia and industry, the issues still remain. In this paper, for resolving the obstacles, we define a new problem, ℓ-cohesive m-ridesharing group (ℓm-CRG) query, which retrieves a cohesive ridesharing group by considering spatial, social, and temporal information. The problem is based on the three underlying assumptions: people tend to rideshare with socially connected friends, people are willing to walk but not too much, and optimization of finding good groups is essential for both drivers and passengers. In our ridesharing framework, queries are processed by efficiently taking geo-social network data into account. For this purpose, we propose an efficient method for processing the queries using a new concept, exact n-friend set, with its efficient update. Moreover, we further improve our method by utilizing inverted timetable (ITT), which grasps crucial time information. Specifically, we devise time-constrained and incremental personalized-proximity search (TIPS). Finally, the performance of the proposed method is evaluated by extensive experiments on several data sets.

INDEX TERMS Geo-social networks, query processing, ridesharing services, spatial databases.

I. INTRODUCTION

With the development of GPS-enabled smart devices, ridesharing has attracted attention as a solution for resolving crucial societal, environmental, and economic problems [1]–[9]. For example, drivers can alleviate traffic congestion by sharing empty seats of their vehicles. Especially, they use ridesharing services to conveniently find ridesharing groups. For example, Uber, which is a pioneer in the ridesharing industry, successfully provides the service through various riding options such as Express Pool [10]. Other good examples are DiDi [11], BlaBlaCar [12], and Lyft [13]. These services have benefited various fields. According to a recent study [14], the potential reduction in traffic jams is as high as 31%. The decrease in carbon dioxide (CO₂) is another major advantage. In addition, ridesharing can reduce the costs of fuel and parking.

Although many people have already taken advantage of ridesharing services, significant challenges remain. One of the problems is the discomfort faced while traveling with strangers, which is aggravated by untrustworthy riders. This is especially important for weak or young people, tourists in an unknown place, and travelers enjoying a long journey. Unfortunately, commercial services still struggle with addressing this problem. In addition, most existing studies of ridesharing have been weak in forming a cohesive group, including a driver and passengers, because they have mainly focused on other factors such as locations of users, driving fare, and the optimized route.

The use of social networks, which for a decade have been investigated in academia and industry, is one of the reasonable approaches for considering the cohesion of ridesharing members [14]–[21]. A graph consisting of vertices and edges is still effective in expressing the closeness between users, meaning an edge between two users represents their friendships in many studies. An increase in the minimum
Another considerable benefit the ridesharing form reducing traffic congestion, whereupon the simplicity of this model makes it more effective by passengers after departure and en route to the destination.

One of the slugging rules is that slugging \([25]\), which is a simple but potent ridesharing model. \[m\text{-Ridesharing Group} \] based on the available timeslot, cohesion of social groups \([22]–[24]\).

However, the research on the problem of cohesion-related ridesharing is in its infancy. For example, user profiles in social networks were used for matching drivers and passengers \([15]\). However, it is insufficient for organizing cohesive groups. Although the study \([17]\) utilized social network data for considering the harmony of ridesharing members, it assumed that riders arrive at a departure point in time or filtered people who cannot be in time out before processing queries. Furthermore, the study only focused on direct friendship (1-hop friends). In other words, its social model could be somewhat restrictive to the passengers and the driver. Another study \([18]\) also considered social network data but used its own ranking function unlike subgraph models for checking the social aspect. Moreover, the study dealt with a problem for finding top-\(k\) taxis based on trip requests in road networks.

In this paper, we propose a problem, \(t\)-Cohesive \(m\)-Ridesharing Group (\(t\)-\(m\)-CRG) query, which retrieves a cohesive ridesharing group based on the available timeslot, location, friendship of the group members, and the number of empty seats. As a ridesharing model for our study, we adopt slugging \([25]\), which is a simple but potent ridesharing model. One of the slugging rules is that \textit{drivers never pick up new passengers after departure and en route to the destination.}

The simplicity of this model makes it more effective by reducing traffic congestion, whereupon \textit{the ridesharing form has become more popular.} Another considerable benefit of slugging is not just cost-saving but to drive on High-Occupancy Vehicle (HOV) lane \([26]\).

\textbf{Remark 1:} Slugging was originally developed for commuters not entrepreneurs. Nowadays, however, some companies (e.g., Uber and BlaBlaCar) adopt the slugging for providing various options to users. Thus, other service providers also could make their own policy on driving fare regardless of our method.

Figure 1(a) depicts an example of slugging in a ridesharing service. For effective ridesharing, passengers wait for the ridesharing request of a driver after enrolling their requests to the ridesharing service system. When a driver wants to take on some passengers, the driver announces the itinerary to the potential passengers by enrolling information about the trip. If the itinerary is the same or similar, they ride a vehicle together at the departure point of the driver. After arriving at a destination, the passengers alight and go to their respective destinations. Figure 1(b) shows each step of \textit{Express Pool}, which is Uber’s latest ridesharing (slugging) model \([10]\).

In the below, each scenario explains the need to use of a social boundary:

\textit{Scenario 1 (Commute With Acquaintances):} In the morning or evening, commuters usually rideshare with colleagues or someone who travels in a similar area or direction. This helps to mitigate traffic congestion during rush hour by decreasing the number of vehicles on the road. Moreover, ridesharing gives them a chance to make friendships. Sometimes, however, some people want to ride with acquaintances or somebody they have already shared their rides.

\textit{Scenario 2 (Road Trip on Trust):} Nowadays, a road trip is one of the wide-spread approaches for travel using automobiles. To save money on the road trip, many people use ridesharing services if directions of their trips are the same or similar. However, some people have concerns over safety when ridesharing. Although many ridesharing services are keen to solve the problem of safety, sufficient measures have certainly not been taken.

The contributions of this paper are as follows:

- We define a new type of ridesharing queries, the \(\ell\)-\(m\)-CRG query, which finds a ridesharing group, with the target for resolving the trust- and comfort-related problems. The query considers the number of empty seats, departure and arrival points, the social relationship of ridesharing members, and their available timeslots, to assist ridesharing services efficiently.
- We propose a new concept of \textit{exact} \(n\)-friend set made up of only vertices connected by the minimum \(n\) edges. We additionally design the update method of the exact \(n\)-friend set using \textit{n-social counting filter}, and an \(\ell\)-\(m\)-CRG query processing method.
- We develop an enhanced query processing method, \textit{time-constrained and incremental personalized-proximity search (TIPS)}, which consists of \textit{cell-time pruning} and \textit{bidirectional cell-time traversing}, using the \textit{inverted timetable (ITT)} containing time information of riders.
- We conduct various experiments on various data sets to evaluate the performance of our methods. Further, through varying query parameters, we demonstrate the performance of the proposed method for supporting ridesharing services.
Section V discusses the performance of our methods, and in Section IV, we propose a further improved method. Section III presents the expected departure and arrival times. For optimizations, we introduce the ridesharing system architecture in Section II-A and a problem definition in Section II-B. Additionally, we discuss a naïve approach to process the ℓm-CRG query in Section II-C. Table 1 summarizes the notations commonly used throughout this paper.

### A. RIDESHARING SERVICE SYSTEM

In our ridesharing system (Figure 2), service users can choose their roles by specifying and sending ridesharing requests. The storage and index of a ridesharing system are updated by an update module based on the requests of passengers. Each request contains the available (preferred) timeslot and itinerary. The search space of a query processor is made by all passengers' requests. Meanwhile, the ℓm-CRG queries are created according to the requests of drivers. Each driver's request includes departure and arrival times, an acceptable social boundary, an itinerary, and the number of available seats. After receiving the requests, the ridesharing system creates and processes a query for matching a driver and passengers optimally. The result is returned to the driver and passengers after processing the query. Concretely, the answer consists of a driver, other passengers, an itinerary, and departure and arrival times. In this paper, we especially focus on organizing a ridesharing group owing to its importance in ridesharing services. Note that the search space is always changed by the requests of passengers, so the size is usually city- or country-scale.

### B. PROBLEM STATEMENT

In this work, we define the ℓm-CRG on the basis of the three underlying assumptions: people tend to rideshare with socially connected friends, people are willing to walk but not too much, and optimization of finding good groups is essential for both drivers and passengers.

Given an undirected social graph $G = \{V, E\}$, a vertex $v (u) \in V$ means a passenger (a driver), i.e., a data object, and an edge $e(v, v') \in E$ denotes the friendship between two data objects $v$ and $v'$. Intuitively, the fewer number of edges indicates a closer friendship. Each $v$ has its own itinerary $p_v = (p_v^{d}, p_v^{a})$, where $p_v^{d}$ and $p_v^{a}$ represent departure and arrival points, and the possible timeslot $t_v = [t_v^{d}, t_v^{a}]$ ($t_v^{d} \leq t_v^{a}$), where $t_v^{d}$ and $t_v^{a}$ denote the earliest possible time of departure and the latest possible time of arrival (in short, departure and arrival times). For simplicity, we set the timeslot to one hour in this paper, but it can always be changed. Without ambiguity, for instance, $[1 : 2]$ indicates a timeslot from 01:00 to 02:59.

To determine group cohesion, we define a new notion of ℓ-cohesive group similar to club [22], which is one of the popular subgroup models in the literature of social network analysis, but slightly relaxed.

**Definition 1 (ℓ-Cohesive Group):** Given an undirected social graph $G = \{V, E\}$, an ℓ-cohesive group is a subgraph whose minimum social diameter (distance) $d_v(v, v')$ is less than or equal to an acceptable social boundary (integer) ℓ ($> 0$), where $v$ and $v'$ are members of the ℓ-cohesive group.

However, all of the ℓ-cohesive groups cannot be candidates of the answer of the ℓm-CRG query because a car has a limited number of seats. In other words, m members of groups can only be the result of ridesharing requests if the number of seats is $m$.

**Definition 2 (ℓ-Cohesive m-Group):** Given an undirected social graph $G = \{V, E\}$ and the number of seats $m$,
every \( \ell \)-cohesive group, which consists of \( m \) members, is an \( \ell \)-cohesive \( m \)-group \( G^{(\ell)}_m = \{v_1, \ldots, v_m\} \).

**Example 1:** Figure 3 depicts an example of \( G^{(\ell)}_m \) (\( \ell = 2, m = 3 \)). Consider a social network that consists of six vertices \( \{u, v_1, \ldots, v_5\} \), six edges \( \{e(u, v_1), e(u, v_2), \ldots, e(v_4, v_5)\} \), and five groups (a)-(e). There are several other groups, but we mainly describe groups including a driver \( u \) because \( u \) must be a member of a result group. A group (a) is composed of (direct or 1-hop) friends \( v_1, v_3, \) and \( u \), and the maximum \( d_u(v, v') \) of groups (b) and (c) is 2 (\( \leq \ell \)). Thus, the three groups can be candidates of the final answer. However, a group (d) cannot be an eventual group because \( v_2 \) is connected to \( v_4 \) through three edges. Even though a group (e) has a proper \( d_u(v, v') \), \( u \) is excluded from this group. The \( u \) has to be included in the final group because \( u \) can only drive a car. Furthermore, \( v_5 \) can never be a member of candidate groups because \( v_5 \) has no acceptable social friendship with \( u \) in this social network.

Not only a driver and passengers but also the passengers of an \( \ell \)-cohesive \( m \)-group are linked together within \( \ell \) edges. One may think extreme cases based on six-degree of separation [27]; e.g., every maximum number of edges between each member is \( \ell \). However, the cases are rare thanks to the property of the group structure. In other words, members of a result group naturally share some popular friends [28], [29]. Furthermore, the uncommon case could be also acceptable especially to someone who wants to travel with slightly trustworthy strangers for privacy concerns. The important thing is that each member must be linked by other friends or friends of a friend. This is why we choose this subgraph model for grouping the driver and passengers in our ridesharing system. Note that we can also form groups consist of only people connected by exact \( \ell \) edges by simply using the exact n-friend set in Section III-A although we do not explain in this paper.

Moreover, available times of drivers and passengers do not always match. Specifically, passengers enroll their time information at the ridesharing service system before the arrival of a driver request. After receiving the ridesharing request, the ridesharing system searches the result of the request by checking whether the available (usually departure and arrival) time of the driver is included by those of passengers.

---

**FIGURE 3.** Example of \( \ell \)-cohesive \( m \)-group (\( \ell = 2, m = 3 \)).

---

**FIGURE 4.** Example of temporal condition.

**Definition 3 (Temporal Condition):** Every passenger \( v_i \) of a result group \( G^{(\ell)}_m \) satisfies the following condition (1 \( \leq i \leq m - 1 \)):

\[
v_i, u \in G^{(\ell)}_m \land d_u^i \leq t_u^i \land t_u^a \leq t_u^i. \tag{1}\]

Throughout the paper, we also use another notation \( t_u \subseteq t_v \) for denoting \( t_u^i \leq t_v^i \) and \( t_u^a \leq t_v^a \) at once. Note that the available timeslot does not mean the fixed departure and arrival times, so group members can make their own plans by considering the travel. The arrival times of drivers are generally decided by the car navigation system they use.

**Example 2:** Figure 4 describes the available timeslots of five passengers and a driver \( u \). Their social network is shown in Figure 3. Passengers \( v_1, v_3 \) and \( v_4 \) can ride a car together because their timeslots fully overlap the timeslot \( t_u = \{11:13\} \) of \( u \). However, the possible departure time of \( v_2 \) is later than that of \( u \), so it is difficult for \( v_2 \) to join the final group. Though \( v_5 \) can arrive at the departure point before 11, his/her arrival time is before that of the driver, i.e., \( t_v^d < t_u^a \) of \( u \). Furthermore, \( s \) has a deficient friendship with others.

In our ridesharing model, passengers may slightly travel from their positions to the departure point of a driver because the driver never picks up or drops off anyone while driving. Intuitively, a longer distance between the locations of a passenger and a driver means an outside chance of successful matching of a ridesharing group. For this purpose, we need to know each intuitive movement cost of a passenger and an \( \ell \)-cohesive \( m \)-group.

**Definition 4 (Movement Cost of a Passenger):** Given a driver \( u \) and a passenger \( v \), the movement cost \( MC(v) \) of \( v \) is defined as:

\[
MC(v) = \sum_{p_u^d, p_v^d} d_{g}(p_u^d, p_v^d) + \sum_{p_u^a, p_v^a} d_{g}(p_u^a, p_v^a), \tag{2}\]

where \( d_{g}(a, b) \) indicates the shortest geographical such as Euclidean and a road network distances between \( a \) and \( b \).

**Definition 5 (Movement Cost of an \( \ell \)-Cohesive \( m \)-Group):** Given an \( \ell \)-cohesive \( m \)-group \( G^{(\ell)}_m \), the movement cost \( MC(G^{(\ell)}_m) \) of \( G^{(\ell)}_m \) is defined as (1 \( \leq i \leq m - 1 \)):

\[
MC(G^{(\ell)}_m) = \sum_{u, v_i \in G^{(\ell)}_m} MC(v_i). \tag{3}\]

The cost is one of the well-known metrics in ridesharing and can be easily substituted for the distance on road networks [17], [19].
Meanwhile, it is unnecessary for passengers to get into a car if their own movement costs are higher than the shortest distance between their departure and arrival points. In other words, our ridesharing system must filter that sort of passenger out because self-travel is better than ridesharing in that case.

**Definition 6 (Spatial Condition):** Spatial condition to be satisfied by every passenger \( v \) is formally presented as \((1 \leq i \leq m - 1)\):

\[
v_i \in G_m^{\ell} \land MC(v_i) < d_s(p_u^d, p_v^d).
\]

As for another spatial aspect, we also define the movement threshold \( h \) to practically apply our methods to real-world services, that is, passengers whose \( MC(v) \) exceeding \( h \) can never be a member of a query answer. For example, system managers or the driver can decide on the value of \( h \) by considering their environments because it should be mostly different by region or time. We experimentally discuss the effect of \( h \) in the section on performance evaluation.

**Definition 7 (\( \ell \)-Cohesive m-Ridesharing Group Query):**

Given \( G = (V, E) \) and an \( \ell \)-cohesive m-ridesharing group query \( q = (u, p_u, t_u, \ell, m) \), where a query user \( u \), itinerary information \( p_u = (p_u^d, p_u^a) \), departure point \( p_u^d \) and arrival point \( p_u^a \), time information \( t_u = (t_u^d : t_u^a) \), departure time \( t_u^d \) and arrival time \( t_u^a \), acceptable social boundary \( \ell \), and the number of available seats \( m \), the \( \ell \)-CRG query finds a result set \( R = \{v_1, v_2, \ldots, v_{m-1}\} \), which is an \( \ell \)-cohesive m-group \( G_m^{\ell} \) mostly minimizing their own movement cost \( MC(G_m^{\ell}) \), where \( t_v \) is within every \( t_u \).

**Example 3:** Figure 5 shows an example of an \( \ell \)-CRG query considering friendship and timeslots in Figures 3 and 4 (\( \ell = 2, m = 3, t_u = \{11 : 13\} \)). First, \( v_1 \) is the first candidate for a result. Even though \( MC(v_1) \) is comparable to \( MC(v_3) \), \( v_5 \) is insufficiently close to \( u \) in the previous (friendship) examples. The next smallest movement cost is \( v_2 \)’s one. However, \( v_2 \) is also filtered out due to the temporal condition. Consequently, the final group of the query is \( \{u, v_1, v_3\} \).

A brute-force approach for processing the \( \ell \)-CRG query is described below. First, we construct all \( G_m^{\ell} \) and calculate every \( MC(G_m^{\ell}) \). Then, whenever calculating \( MC(G_m^{\ell}) \), we compare a current minimum \( MC(G_m^{\ell}) \) with a newly measured \( MC(G_m^{\ell}) \), and store the smaller one to find the least \( MC(G_m^{\ell}) \) and its \( G_m^{\ell} \). However, it might understandably cause some inefficiencies. Specifically, forming all \( G_m^{\ell} \) takes \( O(c_m^{V}) \) time, although this approach finds an exact result. Consequently, an improved method is necessary to tackle such a drawback.

**Remark 2:** Though someone may concern that social network (graph) data may be insufficient for ensuring the safety and comfort perfectly, an increasing number of social-aware ridesharing has been investigated [14]–[21], [30]. One of the main reasons is that it is easy to make a social graph, i.e., friendships of users, without strenuous efforts. Existing systems and services can be equipped with the graph by matching previous grouped members like [16]. Another practical method is to cooperate with other social media they have already joined. Next, lots of valuable works related to the graph have been still proposed, so it will be other chances for the social-aware ridesharing to be improved with the latest graph algorithms.

### C. NAÏVE SOLUTION

A naïve method (called Naive) for processing the \( \ell \)-CRG query utilizes a simple branch and bound technique. The main concept of Naive is to expand its search space constantly until a query answer is found. In other words, we gradually seek two nearest \( p_u^d \) and \( p_u^a \) to \( p_v^d \) and \( p_v^a \) by turns. Note that \( v \) and \( v' \) are probably not always the same.

Given an \( \ell \)-CRG query, we find the nearest \( p_u^d \) to \( p_v^d \). Conceptual partitioning (CPM) [31], which is a commonly used method for processing the nearest neighbor queries, is adopted for incrementally finding those points. In dynamic environments, specifically, CPM conceptually divides a normal grid plane \( C \) into various-sized rectangles, which enclose at least one grid cell \( c \in C \). It helps a query processor in efficiently reducing the number of cell accesses, which is directly related to the performance of the search algorithms.

Whenever searching \( p_v^d \), we check the friendship between a corresponding \( v \) and \( u \) by using a breath-first approach among the standard graph algorithms because a final ridesharing group only consists of some \( v \) such that \( d_s(v, u) \leq \ell \). We also check the spatial and temporal conditions simultaneously. If the checked \( v \) meets these conditions, we insert \( v \) into the checked list of the departure side. Next, the same work is executed at the arrival side by repeatedly retrieving \( p_v^a \). The whole above procedure is repeated until searching a query answer or an entire search space if there is no query result. In the middle of that iteration, we try to create \( G_m^{\ell} \) by checking the social condition among the candidate passengers if the number of \( u \) and \( v \) is over or equal to \( m \). Then, if current \( m - 1 \) passengers satisfy the social condition properly, we calculate \( MC(G_m^{\ell}) \). When there is no chance to make a better group, Naive returns a current \( G_m^{\ell} \) as a query answer by stopping this algorithm before exploring an expected whole area.

### III. THE PROPOSED METHOD

We first propose a novel concept of an exact n-friend set in Section III-A, its update method using n-social counting filter in Section III-B, and incremental personalized-proximity search (IPS) in Section III-C.
A. EXACT n-FRIEND SET

Although we are surely able to find a query answer using Naive, there are some drawbacks. First, when finding \( p^\ell \) after checking the corresponding \( v \) of \( p^\ell \), we can avoid redundant breadth-first search by adopting a notion of \( \ell \)-hop friend list [32]. Especially, this is effective in checking the social relationship among the passengers, i.e., \( G_m^{\ell} \).

**Definition 8 (\( \ell \)-Hop Friend List):** \( V_v^{(\ell)} \subset V \) denotes the \( \ell \)-hop friend list of \( v \) such that:

\[
V_v^{(\ell)} = \begin{cases} \{ v' \mid \exists e(v, v') \in E \} & (\ell = 1), \\ V_v^{(\ell-1)} \cup \{ v' \mid \exists e(v', v'') \in E, v'' \in V_v^{(\ell-1)} \} & (\ell > 1). \end{cases}
\]

(5)

However, the search performance, which highly depends on the method for creating each \( V_v^{(\ell)} \), could be somewhat inefficient when \( \ell \) increases. That is, there is room for boosting the performance by improving the simple method for creating \( V_v^{(\ell)} \). For the enhancement, we propose a new concept of a set \( F_v^{(n)} \) of \( v \)'s exact \( n \)-friends only connected with \( v \) by the minimum \( n \) edges.

**Definition 9 (Exact n-Friend Set):** Exact \( n \)-friend set \( F_v^{(n)} \subset V \) of \( v \) is the element of an \( n \)-hop friend list except (\( n \)-1)-hop friend list and formally presented as:

\[
F_v^{(n)} = \begin{cases} v, & (n = 0), \\ V_v^{(n)} \setminus V_v^{(n-1)}, & (n > 0). \end{cases}
\]

(6)

Every \( F_v^{(n)} \) must be maintained by every \( v \) prior to the query processing. When propagating friend lists, we combine several sets, e.g., \( F_v^{(1)} \) and \( F_v^{(n)} \), by avoiding redundant graph traversing and check of social connections. If the query input \( \ell \) is over \( n \), we just start graph search from \( V_v^{(n)} \). Hence, the preprocessing of \( F_v^{(n)} \) undoubtedly reduces the time of query processing. However, a simple update method of \( F_v^{(n)} \) might be time-consuming, which is definitely one of the main weaknesses. For dealing with the issue, an efficient update method is proposed in Section III-B. Note that the number \( n \) of \( F_v^{(n)} \) is flexible according to the preference of system managers or hardware capabilities.

B. UPDATE OF EXACT n-FRIEND SET USING n-SOCIAL COUNTING FILTER

A major downside of the exact \( n \)-friend set is a feeble update. When some users want to break social connections or opt out of the service, their friend lists have to change or are removed. Then, all the exact \( n \)-friend sets of relevant vertices also need to be updated. However, a basic method for looking each friend list up is definitely inefficient. Especially, the performance of the update grades badly when the size of friends changed is bigger.

For tackling the issue, we devise an \( n \)-social counting filter \( S_v^{(n)} \), which grasps information about how many edges are connected with each vertex in the exact \( n \)-friend set. Then we can decide whether exact \( n \)-friend sets must be revised or not. The gist is that we do not always modify all exact \( n \)-friend sets if there are still other connections.

![Diagram of Exact 2-Friend Sets and 2-Social Counting Filters of \( v_1 \) and \( v_3 \).](image)

**Definition 10 (n-Social Counting Filter):** The \( n \)-social counting filter \( S_v^{(n)} \) is a data structure composed of a set of mapping functions \( S_v^{(n)}(v) \) to the numbers of edges between \( F_v^{(n-1)} \) and \( F_v^{(n)} \) such that \( (0 \leq S_v^{(n)}(v') \leq |F_v^{(n-1)}|) \):

\[
S_v^{(n)}(v') = |\{ e(v', v'') \mid v' \in F_v^{(n)}, v'' \in F_v^{(n-1)} \}| \quad (0 < n).
\]

(7)

An integer \( n \) is chosen by the \( n \) used in the exact \( n \)-friend set. Furthermore, \( S_v^{(n)} \) has to be created when making \( F_v^{(n)} \) simultaneously.

**Example 4:** Figure 6 depicts an example of a 2-social counting filter. The 2-social counting filters of \( v_1 \) and \( v_3 \) are shown in Figure 6(a). First, consider \( V_{v_1}^{(1)} = F_{v_1}^{(1)} = \{ v_2, v_3, v_4, v_5 \} \) and \( F_{v_1}^{(2)} = \{ v_6, v_7, v_8 \} \). Then \( S_{v_1}^{(2)}(v_6) = 3 \) because the number of edges between \( V_{v_1}^{(1)} \) and \( v_6 \) is three. The \( S_{v_3}^{(2)} \) is created in the same manner according to \( V_{v_3}^{(1)} = F_{v_3}^{(1)} = \{ v_1, v_4, v_6, v_8 \} \), and \( F_{v_3}^{(2)} = \{ v_2, v_5, v_7, v_9 \} \). At the moment, suppose an edge \( e(v_1, v_3) \) is removed and the scenario is presented in Figure 6(b). If \( v_1 \) has only a friend \( v_3 \), we would get rid of \( v_6 \) and \( v_8 \) from \( F_{v_1}^{(2)} \). In this case, however, \( v_1 \) has other friends such as \( v_2, v_4, v_5, v_7, v_9 \), so \( v_6 \) could remain in \( F_{v_1}^{(2)} \) exactly thanks to \( e(v_2, v_6) \) and \( e(v_4, v_6) \). On the other hand, we have to delete \( v_8 \) from \( F_{v_3}^{(2)} \) because there is no other edges between \( V_{v_1}^{(1)} \) and \( v_8 \). We easily figure out the number of connected edges using \( S_{v_3}^{(2)} \) instead of the traversing all friends \( V_{v_1}^{(1)} \) of \( v_1 \). Furthermore, it is necessary to modify not only \( F_{v_1}^{(2)} \) but also \( F_{v_3}^{(2)} \) owing to the deletion of \( e(v_1, v_3) \). Someone might think that \( v_3 \) and \( v_4 \) are the entries of \( F_{v_1}^{(2)} \) and \( F_{v_3}^{(2)} \), and \( v_6, v_7, v_9 \) and \( v_8 \) are the members of \( V_{v_1}^{(1)} \) or \( V_{v_3}^{(1)} \) but that is wrong because \( \ell \) or \( n \) is decided by the minimum number of edges according to Definition 8.

Algorithm 1 describes a method for updating the friend list with the exact \( n \)-friend set. When receiving an update command \( udl \), a function \( UpdateFriList \) is executed for.
deleting or inserting an edge. A simple update of the social graph just modifies \( V_{1} \) and \( V_{v} \) in terms of \( e(v, v') \), but we have to invoke UpdateEnS twice per \( u \) to update \( S^{(n)} \) as well.

The UpdateEnS detailed in Algorithm 2 is mainly divided by two functions DeleteSCF and InsertSCF. Although we just consider the DeleteSCF when the \( u \) is the deletion of \( e(v, v') \), the insertion of \( e(v, v') \) could bring a complex process because we need to check whether \( v' \) is already linked with \( v \) through some edges or not. According to Definitions 8 and 9, the \( v' \) cannot be contained in more than one group such as \( V_{1} \), and \( V_{0}^{(2)} \). If so, we remove \( v' \) from a related group before invoking the InsertSCF (Lines 1-13). We recursively traverse \( F_{1}^{(n)} \) in DeleteSCF and InsertSCF. At this point \( S^{(0)}_{v} \) and \( S^{(0)}_{v'} \) (\( f \in F_{v}' \)) are modified. Specifically, whenever each \( f \) is checked, we subtract 1 from \( S^{(0)}_{v} \) and \( S^{(0)}_{v'} \). Furthermore, we delete \( f \) from \( F_{v}' \) and \( v \) from \( F_{v}' \), if the checked \( S^{(0)}_{v}(f) \) (or \( S^{(0)}_{v'}(v) \)) is zero, which means the minimum number of edges between \( v \) and \( f \) is not 1 anymore (Lines 14-24). On the other hand, a similar process is executed when the \( u \) is the insertion of \( e(v, v') \) by considering the revision of \( S^{(0)}_{v} \) and \( F^{(0)} \). After inserting \( f \) and \( v \) into \( F_{v}^{(1)} \) and \( F_{v'}^{(1)} \), we particularly add 1 to \( S^{(1)}_{v}(f) \) and \( S^{(1)}_{v'}(v) \) as regards every \( f \) (Lines 25-35).

C. INCREMENTAL PERSONALIZED-PROXIMITY SEARCH (IPS)

We introduce the IPS, a novel method for processing \( \ell m \)-CRG query. Given an \( \ell m \)-CRG query, we incrementally seek two nearest \( c \) and \( c' \) covering \( p_{v}^{d} \) and \( p_{v'}^{d} \), where \( v, v' \in V_{u}^{(1)} \). That is, we do not need to examine some cells, which do not cover at least one \( p_{v}^{d} \) or \( p_{v'}^{d} \) of \( v \in V_{u}^{(1)} \), because \( V_{u}^{(1)} \) contains all members of a final ridesharing group. Thus, the incremental finding of only the nearest \( v \in V_{u}^{(1)} \) is a crucial issue. For the purpose, we efficiently construct \( V_{u}^{(1)} \) by exploiting \( V_{u}^{(d)} \) and \( F_{u}^{(n)} \). Then, we examine every \( v \) of \( p_{v}^{d} \) in \( c \) and check whether \( v \) is in \( V_{u}^{(1)} \) and satisfies the spatial and temporal conditions. If so, \( v \) is one of the candidates of the final group members. Similarly, we conduct an identical search at the opposite side, i.e., \( v' \) of \( p_{v'}^{d} \) in \( c' \). When the number of sum of those \( v, v' \), and \( u \) is over or equal to \( m \), we try to create \( G_{m}^{(d)} \) by using each \( \ell \)-hop friend list and exact \( n \)-friend set. After that, whenever a new \( p_{v}^{d} \) or \( p_{v'}^{d} \) is found, we attempt to construct a new \( G_{m}^{(d)} \) and check whether the group can be a result group. This procedure continues until ensuring that there is no answer or a current group is optimal.
covering at least one departure or arrival point of a member of \( \mathcal{V} \), we invoke a function checking the size of \( \mathcal{F} \). MakeFriendList unconditionally contains \( \mathcal{V} \). Although \( \mathcal{V} \) 's timeslots always satisfy that of the driver for simplicity.

Example 5: Figure 7 shows an example of \( \ell\)-CRG query processing. Consider a driver \( u \), eight passengers = \( \{v_1, \ldots, v_8\} \), their social network, and its iteration table with candidates and movement costs. Furthermore, a grid, composed of 105 cells, covers their itineraries. Suppose passengers’ timeslots always satisfy that of the driver for simplicity. When processing an \( \ell\)-CRG query (\( \ell = 2, m = 3 \)), we first check a cell \( c_{88} \) covering \( p_{11} \) because it is the nearest one among cells covering departure locations of \( \ell\)-hop friends of \( u \). Although \( p_{11} \) is the nearest point to \( p_{11} \), we do not need to search \( c_{88} \) because \( v_4 \) is not a member of \( \mathcal{V}_u \). Thus, \( v_1 \) with \( MC(v_1) \) is inserted into a candidate list ordered by \( MC(v) \). Next, we search \( c_{88} \) as a check of an arrival side, and then insert \( v_2 \) into the candidate list. The first element in the candidate list is now \( v_2 \) not \( v_1 \) because \( MC(v_2) < MC(v_1) \). The above procedure continues until inserting \( v_6 \) into the list. Two points \( p_{17} \) and \( p_{19} \) are the next order but we can terminate this algorithm before searching \( c_6 \) and \( c_{25} \) because \( MC(v_2) < d_4(p_{17}, c_6) + d_4(p_{19}, c_{25}) \) (Theorem 1). In conclusion, we have searched only shaded cells, and a final answer is \( \{u, v_2, v_6\} \).

Although the incremental manner of this method is similar to that of Naive, we can further avoid several unnecessary computations and data accesses through the proposed approach. We describe a pseudocode of the query processing method in Algorithm 3. First, initialize the basic components. Then, we construct \( V_u^{(i)} \) to verify the \( v \) of incrementally retrieved \( p_v \) or \( p_v' \) because a query answer unconditionally contains \( m - 1 \) passengers in \( V_u^{(i)} \). A function MakeFriendList constructs \( V_u^{(i)} \) by exploiting \( V_u^{(i)} \) and \( F_u^{(i)} \) (Lines 1-4). Next, we can prevent worthless jobs by checking the size of |\( V_u^{(i)} \cup u \)|. If its size is over or equal to \( m \), we invoke a function FindCoverCell to insert all cells covering at least one departure or arrival point of a member of 

### Algorithm 3 \( \ell\)-Cohesive \( m\)-Ridesharing Group Query Processing Using Exact \( n\)-Friend Set

| Input | Output |
|-------|--------|
| \( \ell\)-CRG query \( q = (a, p_v, t_u, \ell, m) \) | result set \( \mathcal{R} \) |

**Procedure:**
1. Set result set \( \mathcal{R} \leftarrow \emptyset \), checked set \( \mathcal{C} \leftarrow \emptyset \), intermediate set \( \mathcal{S} \leftarrow \emptyset \);
2. Set \( d_{1, \ell}\)-minimized high-max \( \mathcal{DH} \leftarrow \emptyset \), \( \mathcal{AH} \leftarrow \emptyset \), \( \mathcal{MC}(v)\)-ordered min-max \( \mathcal{VH} \leftarrow \emptyset \), \( \mathcal{MC}(v)\)-ordered candidate list \( \mathcal{CL} \leftarrow \emptyset \);
3. Set bestCost \( \leftarrow \infty \), candiCost \( \leftarrow -\infty \);
4. \( V_u^{(i)} \) \leftarrow MakeFriendList \((u, \ell)\);
5. if \( m \leq |V_u^{(i)} \cup u| \)
   6. \( (\mathcal{DH}, \mathcal{AH}) \leftarrow FindCoverCell \((u, V_u^{(i)}, \mathcal{DH}, \mathcal{AH})\); 
   7. while \( \mathcal{DH} \neq \emptyset \) or \( \mathcal{AH} \neq \emptyset \)
      8. \( c \leftarrow \emptyset \mathcal{DH} \);  
      9. \( c' \leftarrow \emptyset \mathcal{AH} \); 
      10. if bestCost \( \leq \) candiCost \( \leq MC(v) \)
          11. break;  // Theorem 1 
      12. \( \mathcal{VH} \leftarrow SearchCellData \((c, \mathcal{VH}, \mathcal{CS}, V_u^{(i)}\); 
      13. \( \mathcal{VH} \leftarrow SearchCellData \((c', \mathcal{VH}, \mathcal{CS}, V_u^{(i)}\);
      14. while \( \mathcal{VH} \neq \emptyset \)
         15. \( v \leftarrow \emptyset \mathcal{VH} ; 
         16. if bestCost \( \leq \) candiCost \( \leq MC(v) \)
            17. break;  // Lemma 1 
         18. \( \mathcal{I} \leftarrow ComputeGroup \((v, \ell, m, \mathcal{IS}, \mathcal{CL})\); 
         19. if \( MC(IS) \leq \) bestCost
            20. \( \mathcal{R} \leftarrow \mathcal{IS} ; 
            21. bestCost \leftarrow MC(IS) ; 
            22. candiCost \leftarrow \sum_{i=1}^{c-2} (MC(v)^{ith} in \mathcal{CL}); 
            23. return \( \mathcal{R} ; 
            24. Function \( FindCoverCell \((u, V_u^{(i)}, \mathcal{DH}, \mathcal{AH})\,: 
               25. for each \( v \in V_u^{(i)} \)
                  26. \( c \leftarrow \) cell covering \( p_v \); 
                  27. \( c' \leftarrow \) cell covering \( p_v' \); 
                  28. if \( c \neq \mathcal{DH} \)
                     29. \( \mathcal{C} \leftarrow \) Insert \( c \) into \( \mathcal{DH} \); 
                     30. if \( c' \neq \mathcal{AH} \)
                        31. \( \mathcal{C} \leftarrow \) Insert \( c' \) into \( \mathcal{AH} \); 
               32. return \( \mathcal{DH}, \mathcal{AH} \); 
               33. Function \( SearchCellData \((c, \mathcal{VH}, \mathcal{CS}, V_u^{(i)}\); 
                  34. for each \( v \in p_v \) in \( c \) 
                     35. if \( v \notin \mathcal{CS} \)
                        36. \( \mathcal{IS} \leftarrow \) Insert \( v \) into \( \mathcal{CS} \); 
                        37. else 
                           38. if \( MC(v) < d_4(p_v, p_v') \) and \( t_v \leq t_i \) and \( v \in V_u^{(i)} \)
                              39. \( \mathcal{IS} \leftarrow \) Insert \( v \) into \( \mathcal{VH} \); 
                  40. return \( \mathcal{VH}, \mathcal{CS} \); 
                  41. Function \( ComputeGroup \((v, \ell, m, \mathcal{IS}, \mathcal{CL})\); 
                     42. \( \mathcal{IS} \leftarrow \emptyset ; 
                     43. Insert \( v \) into \( \mathcal{CL} \); 
                     44. \( V_u^{(i)} \) \leftarrow MakeFriendList \((v, \ell)\); 
                     45. if \( m \leq |\mathcal{CL} \cup u| \)
                        46. \( \mathcal{IS} \leftarrow \) FindBestGroup \((\mathcal{CL})\); 
                        47. return \( \mathcal{IS}, \mathcal{CL} \); 

![Diagram of social network](image-url)
$V^d_c$ into the min-heaps $D\mathcal{H}$ and $A\mathcal{H}$, respectively (Lines 5-6). That is, the min-heaps absolutely contain all cells related to candidate passengers after finishing $\text{FindCoverCell}$. Subsequently, this algorithm continues until two heaps have no cell because empty heaps mean that we have checked all cells or data objects. If heaps are not vacant, two $c$ and $c'$ are first popped from $D\mathcal{H}$ and $A\mathcal{H}$. We can terminate this algorithm if the gap between $\text{bestCost}$ and $\text{candiCost}$ is greater than or equal to the sum of $d_g(p^d_u, c)$ and $d_g(p^d_v, c')$ (Lines 7-11). This is proved by Theorem 1 at the end of this section. The above condition can never be satisfied at the first trial because both $\text{bestCost}$ and $\text{candiCost}$ are initialized to infinite numbers. Thus, we search all $p^d_u$ and $p^d_v$ in $c$ and $c'$ by invoking a function $\text{SearchCellData}$. From that, we can obtain a min-heap $\mathcal{V}H$ containing some passengers $(e \in V^d_c)$, which meet the temporal and spatial conditions (Lines 12-13). If $\mathcal{V}H$ has at least one entry, we repeatedly pop $v$, which has the minimum $MC(v)$ from $\mathcal{V}H$ until $\mathcal{V}H$ is emptied. Otherwise, we can terminate this iteration by checking whether $MC(v)$ is greater than or equal to the gap between $\text{bestCost}$ and $\text{candiCost}$. This is also proved by the following Lemma 1. If it is satisfied, this iteration can be stopped because we ensure that a current $\mathcal{V}H$ contains no candidate $v$ having better (lower) $MC(v)$. If not, we invoke a function $\text{ComputeGroup}$ that tries to organize a candidate group by checking the relationships using each $V^d_c$ (Lines 14-18). Specifically, a function $\text{FindBestGroup}$ in the ComputeGroup checks whether the only just-popped $v$, $u$, and other vertices in a candidate list can be $G^d_m$ or not. Next, if a current bestCost is greater than $MC(S)\mathcal{V}A$, we update $\mathcal{R}S$ and bestCost. Then, we set $\text{candiCost}$ as the minimum sum of movement costs of the first $m - 2$ vertices in $MC(v)$-ordered min-heap $\mathcal{C}L$ (Lines 19-23). We repeat this procedure until $D\mathcal{H}$ and $A\mathcal{H}$ are empty or Theorem 1 is satisfied. We finally return $\mathcal{R}S$ as a final answer to the $\ell m$-CRG query.

**Lemma 1:** Let $\text{bestCost}$ be the movement cost of a current optimal $\ell$-cohesive $m$-group including $u$, and $\text{candiCost}$ be the minimum sum of movement costs of $m - 2$ vertices in $MC(v)$-ordered min-heap $\mathcal{C}L$. Then, suppose that $v_j$ is just popped from $MC(v)$-ordered min-heap $\mathcal{V}H$. At this time, if $\text{bestCost} - \text{candiCost} \leq MC(v_j)$, there is no vertex which can make the better $\ell$-cohesive $m$-group in $\mathcal{V}H$.

**Proof:** If $u$ and the first $m - 1$ vertices in $\mathcal{C}L$ are $G^d_m$, its $MC(G^d_m)$ is the current minimum movement cost of an optimal $\ell$-cohesive $m$-group. However, the first $m - 1$ vertices in $\mathcal{C}L$ cannot always be members of $G^d_m$. So $\text{bestCost}$ is greater than or equal to $MC(G^d_m)$. Then $\text{bestCost}$ never change after satisfying that $\text{bestCost} \leq \text{candiCost} + MC(v_j)$, because the next popped $v$ from $\mathcal{V}H$ has definitely the larger movement cost. Hence, Lemma 1 holds.

**Theorem 1:** Let $\text{bestCost}$ and $\text{candiCost}$ be those of Lemma 1. Additionally, suppose two cells $c$ and $c'$ are just popped from $D\mathcal{H}$ and $A\mathcal{H}$. Algorithm 3 can terminate if $\text{bestCost} - \text{candiCost} \leq d_g(p^d_u, c) + d_g(p^d_v, c')$.

Proof: Suppose $p^d_u$ and $p^d_v$ are the next nearest points covered by $c$ and $c'$. Then, we have

$$
\begin{align*}
&d_g(p^d_u, c) \leq d_g(p^d_u, p^d_v), \\
&d_g(p^d_u, c') \leq d_g(p^d_v, p^d_v') \quad (p^d_v \text{ in } c \land p^d_v' \text{ in } c').
\end{align*}
$$

Since $d_g(p^d_u, p^d_v) + d_g(p^d_v, p^d_v') \leq \min\{MC(v), MC(v')\}$, we derive $\text{bestCost} - \text{candiCost} \leq \min\{MC(v), MC(v')\}$. It means that there is no more group which has a lower $MC(G^d_m)$ according to Lemma 1.

**IV. OPTIMIZATIONS**

In Section IV-A, we suggest $\text{ITT}$ for improving the performance of our method. Furthermore, $\text{TIPS}$, an enhanced query processing method, is proposed with cell-time pruning and bidirectional cell-time traversing in Section IV-B.

**A. INVERTED TIMETABLE**

We propose the $\text{ITT}$ for processing $\ell m$-CRG queries efficiently. Although our previous method could efficiently deal with the queries, there are other chances to support the performance of our method because it does not consider time information, which is one of the key roles in ridesharing services. In other words, it is essential to handle time information as well as location and social aspects in our ridesharing system.

When indexing all points $p^d_u$ and $p^d_v$, we also construct each $\text{ITT}_c \in \text{ITT}$ of a cell $c$ considering the time factors of each $v$, i.e., $t^d_v$ and $t^a_v$. Therefore, each $c$ has its own $\text{ITT}_c$ after indexing all data objects. The $\text{ITT}_c$ consists of 24 rows representing 24 hours, and each row is composed of three attributes (fields): ① Time $T$, ② Set $V^d_c(T)$ of departing $v$, ③ Set $V^a_c(T)$ of arriving $v$. Each $v$ is mapped to the corresponding $V^d_c(T)$ or $V^a_c(T)$ by checking $t^d_v$ and $t^a_v$. We define $\text{ITT}$ as follows:

**Definition 11 (Inverted Timetable):** Every cell $c$ has its own $\text{ITT}_c$, where its row consists of unique time $T$ and vertex sets $V^d_c(T)$ and $V^a_c(T)$ such that

$$
\{ V^d_c(T) = \{ v \mid t^d_v = T \}, \} \quad \{ V^a_c(T) = \{ v \mid t^a_v = T \}, \} \quad (0 \leq T \leq 23).
$$

where elements of column $T$ are ranged in ascending order. Note that the column $T$ can be divided into not only 24 but also any other number of rows according to the various system environments.

**Example 6:** Figure 8 displays an example of the inverted timetable. Consider a cell $c_1$ covering two departure points = $\{p^d_{v_1}, p^d_{v_2}\}$ and three arrival points = $\{p^a_{v_1}, p^a_{v_2}, p^a_{v_3}\}$, and the corresponding departure/arrival times in Figure 8(a). According to Definition 11, $V^d_{c_1}(1) = \{v_1\}$ and $V^d_{c_1}(22) = \{v_2\}$. Similarly, $V^a_{c_1}(1) = \{v_3\}$ and $V^a_{c_1}(23) = \{v_1, v_2\}$. The other $V^d_{c_1}(T)$ and $V^a_{c_1}(T)$ are empty. We can see the completed $\text{ITT}_{c_1}$ in Figure 8(b).

The update of $\text{ITT}$ proceeds while conducting that of the location of passengers, whereupon the entire steps are similar to the method in a normal grid index. The difference lies
only in dealing with the time information, which is easily considered.

B. TIME-CONSTRAINED IPS (TIPS)

In this part, we present two types of strategies to prune unnecessary search, and an efficient query processing method taking advantage of the ITT.

1) CELL-TIME PRUNING

We can effectively prune the specific cells if it is guaranteed that the cells do not cover the departure or arrival points of passengers who satisfy the temporal condition (Definition 3). We know whether a cell can be pruned or not by just comparing \( t_u^d \) (of a driver \( u \)) and the minimum \( t_u^d \) (of a passenger \( v \) when \( p_v^d \) is covered by \( c \)). This filters the cases out when there is no \( v \) satisfying the temporal condition, namely the minimum \( t_u^d \) is greater than \( t_u^d \). Thus, we do not need to search this \( c \). The case of the arrival time is similar. If the maximum \( t_u^d \) is lower than \( t_u^d \) when \( p_v^d \) in \( c \), the search about the arrival side is also unnecessary. We formally describe the above concept using the cell-timeslot.

**Definition 12 (Cell-Timeslot):** \( t_c = [t_u^d : t_u^d] \) denotes the cell-timeslot, where \( t_u^d \) is a departure cell-time and \( t_u^d \) is an arrival cell-time such that

\[
\begin{align*}
  t_u^d &= \min(T) \land V_c^d(T) \neq \emptyset, & (0 \leq T \leq 23), \\
  t_u^d &= \max(T) \land V_c^d(T) \neq \emptyset.
\end{align*}
\]

To summarize, given an \( \ell m \)-CRG query, we can skip the search of cells if \( t_u^d < t_u^d \) while processing the departure-side cell search. The arrival-side cell search can also be omitted if \( t_u^d < t_u^d \).

2) BIDIRECTIONAL CELL-TIME TRAVERSING

We traverse a set of passengers if a cell covers some departure or arrival points of the passengers satisfying each temporal condition. At this moment, we are able to efficiently traverse the passengers by utilizing the inverted timetable of the cell. When traversing, for example, passengers whose departure points \( p_v^d \) are in a cell \( c \), we only have to check the departing \( v \) sets \( V_c^d(T) \), where \( t_v^d \leq T \leq t_u^d \). The others in \( V^d(T) \), i.e., \( t_u^d < T \), are intuitively pruned because the elements of the column \( T \) of ITT \( c \) are ranged in ascending order of time. For searching arrival-side passengers, we explore \( V^d(T) \) in reverse, i.e., for \( T \), from \( t_u^d \) to \( t_u^d \).

**Example 7:** Consider the inverted timetable as shown in Figure 8. When processing an \( \ell m \)-CRG query, if the current search is for the departure-side, we traverse the table from \( V_c^d(1) \) to \( V_c^d(t_u^d) \) because of \( t_v^d = 0 \). On the other hand, the search starts from \( V_c^d(23) \) (to \( V_c^d(t_u^d) \)) if the current procedure is for the arrival side. By doing so, we can avoid irrelevant computation, i.e., a set of \( V_c^d(T) \) and \( V_c^d(T) \) if \( t_u^d < T < t_u^d \).

3) QUERY PROCESSING

We propose the TIPS as a final method for processing \( \ell m \)-CRG queries. The above strategies, i.e., cell-time pruning and bidirectional cell-time traversing, should be applied to the query processing method before traversing passengers of each cell in Algorithm 3. Thus, we just convert the function SearchCellData into a function SearchCellTimeData for ease in the explanation of the TIPS.

**Algorithm 4 Time-Aware Cell Search Using Inverted Timetable**

1. **Function** SearchCellTimeData \((c, VH, CS, V^d(T))\):

   2. if \( c \) is from \( DH \) then
      3. Set time \( T \leftarrow t_u^d \);
      4. while \( T \leq t_u^d \) do
         5. for each \( v \in V_c^d(T) \) do
            6. if \( v \notin CS \) then
               7. Insert \( v \) into \( CS \);
            8. else
               9. if \( MC(v) < d_v(p_v^d, p_v^d) \) and \( t_u^d \leq t_u^d \) and \( v \in V_u^d(T) \) then
                  10. Insert \( v \) into \( VH \);
            11. \( T \leftarrow T + 1 \);
         12. else // \( c \) is from \( AH \)
            13. \( T \leftarrow t_u^d \);
            14. while \( t_u^d \leq T \) do
               15. for each \( v \in V_c^d(T) \) do
                  16. if \( v \notin CS \) then
                     17. Insert \( v \) into \( CS \);
                  18. else
                     19. if \( MC(v) < d_v(p_v^d, p_v^d) \) and \( t_u^d \leq t_u^d \) and \( v \in V_u^d(T) \) then
                        20. Insert \( v \) into \( VH \);
               21. \( T \leftarrow T - 1 \);
      22. return \((VH, CS)\);
Algorithm 4 presents the SearchCellTimeData. First, we check whether a current cell \( c \) is from \( \mathcal{DH} \) or \( \mathcal{AH} \). If \( c \) is popped from \( \mathcal{DH} \), we set \( T \) as a departure cell-time \( t_u^d \) (Lines 2-3). We can prune this search if the \( T \) is greater than \( t_u^d \). If not, using a checked set \( C_S \), each \( v \in V_c^d(T) \) is examined for checking that we have already seen the \( v \), i.e., at the arrival-side search (Lines 4-6). When ensuring that this \( v \) is not yet examined, we insert the \( v \) into \( C_S \). Otherwise, we again check whether the \( v \) meets other conditions to be \( G_m^{(t)} \). We insert \( v \) into \( \mathcal{VH} \) if the above conditions are satisfied by the \( v \), and continue the above procedure until \( T \) is equal to the \( u \)'s departure time \( t_u^d \) (Lines 7-11). Finally, we return the completed \( (\mathcal{VH},C_S) \). Arrival-side search is also proceeded in a similar manner, except that we decrease \( T \) from \( t_u^d \) to \( t_u^a \) (Lines 12-21).

V. EXPERIMENTAL EVALUATION

In this section, we describe data sets and basic settings for experiments in Section V-A. Then, we report performance evaluations through extensive experiments: query processing time, area of search space, group matching rate, update time, and voluminous data sets in from Section V-B to Section V-F.

### A. DATA SETS AND SETTINGS

We used four types of data sets, namely \( BE \), \( GE \), \( BA \), and \( GA \). All data sets were sampled from real data sets, Brightkite and Gowalla [33]. Specifically, both \( BE \) and \( GE \) denote data sets in the area of Europe extracted from original data sets (e.g., see Figure 9; an edge connects departure and arrival points). The space of every data set has been normalized to a square of 100 on a side. Additionally, \( BA \) and \( GA \) represent data sets from the area of America. We regard the two points among the check-ins of each vertex as the departure and arrival points. Furthermore, the available departure and arrival times. Table 2 details the real data set statistics.

### TABLE 2. Real data set statistics.

| Data set | # of vertex | # of edge | Avg. # of friend |
|----------|-------------|-----------|------------------|
| BE       | 11,036      | 31,001    | 5.62             |
| GE       | 38,983      | 148,882   | 7.64             |
| BA       | 32,228      | 118,145   | 7.33             |
| GA       | 49,613      | 232,697   | 9.38             |

We next evaluate the performance of our methods. All experiments were conducted on an Intel Core i5 with 3.40 GHz CPU and 16 GB memory, and implemented in Python. We maintained data sets in main-memory, and stored locations of data objects in an index structure of 250 \( \times \) 250 size. We measured the average performance through 200 experiments with random queries. We particularly evaluate our methods by varying four types of parameters. Table 3 lists the parameter settings.

### TABLE 3. Parameter settings.

| Parameter | Setting (default in bold) |
|-----------|---------------------------|
| Social boundary \( \ell \) | 1, 2, (5), 4, 5 |
| Number of seats \( m \) | 2, 3, (4), 5, 6 |
| Movement threshold \( h \) | \((d_g(p_u^d, p_v^a))\), 1, 3, 5, 10, \( \infty \) |
| Driving timeslot \( t_u \) | \([t_u^d: t_u^a]\), [0:1], [2:3], \ldots, [22:23] |

We took its insight of a similar geo-social ridesharing [17]. We took its additional information is in Section VI-C and Appendixes.

### FIGURE 9. Distribution of real data sets.

### FIGURE 10. Average number of \( \ell \)-hop friends varying \( \ell \).
**TIPS** - This method in Section IV-B3 is our final method, which is the same as IPS except cell-time pruning and bidirectional cell-time traversing. Hence this method can effectively avoid irrelevant graph traversing and unnecessary cell searches. Note that we could not plot the result of SER in every single graph because the gap of performances between other methods is significant. Alternatively, we compare the CPU time in Table 4. To tell the result in advance, other algorithms definitely outweigh SER in our problem.

**B. EXPERIMENT FOR QUERY PROCESSING TIME**

We report the running time of experiments with various parameters and data sets in Figure 11. Firstly, Figure 11(a) depicts the effect of varying acceptable social boundary $\ell$ on $BE$. Although the processing times of the three methods increase with $\ell$, TIPS always outperforms other methods because TIPS avoids unnecessary cell search and $\ell$-hop friend check effectively. These tendencies are nearly identical to those of the tests conducted on $GE$, $BA$, and $GA$. According to the above results, the overall performances of the methods heavily depend on varying $\ell$.

Next, we plot the effect of varying empty seats $m$ on $BE$ in Figure 11(b). The performance of TIPS is better than those of the others like the previous experiment. In addition, when $m$ grows, the performance of all methods degrades because larger $m$ obviously makes the methods struggle to search more departure and arrival points.

Third, Figure 11(c) reports the effect of the varying movement threshold $h$ on $BE$. Sometimes potential passengers may not be willing to have long trips from their points to a departure point in real-world services. In this case, drivers are able to specify $h$ by considering their situations. For applying this concept, we can just filter those kinds of passengers whose movement costs are higher than $h$ out when each algorithm
checks other conditions (e.g., spatial and social conditions). Thus, we evaluate the proposed methods by varying $h$ from 1% of the entire width to the infinity. Note that the default $h$ of the remaining experiments is each driver’s own $d_g(p_d, p_u)$. In the experiment, the best algorithm is TIPS and the worst thing is NCF as well. When $h$ increases, the elapsed time decreases because all methods can find an optimal group earlier. In other words, there are a number of people finding it difficult to reach the departure location of a driver on time. One more interesting thing is when there is no limit with regard to $h$, i.e., infinity. Contrary to the expectation, the query processing is fairly time-consuming because we have to consider a greater part of vertices whenever candidate groups are made and checked. Surprisingly, TIPS is stable and it means that lots of passengers cannot meet the time constraints of $u$.

Lastly, Figure 11(d) explains experiments conducted with varying timeslot $t_u$ on BE. Until now, we have described all experiments with each driver’s own timeslot. In other words, the default of $t_u$ is movable. On some occasions, however, the driving time has to be always fixed, for example, school buses. This is why we conducted these tests through classification into 12 timeslots. Then, we can fully catch basic trends even though the time distribution does not always cover all cases. As we expected, the best method is TIPS and the worst method is NCF like other experiments. Furthermore, we can see that all methods take a much longer time at [00:01], [20:21], and [22:23]. One of the reasons is that many people do not log their locations at that time, whereupon our algorithms spent much time for finding optimal groups or terminating their procedures.

We conducted the same tests on GE, BA and GA in Figures 11(e)-(p). All trends are similar to the tests on BE but the average time for processing queries is larger due to the increase of data size. Consequently, the gap would be remarkable as the data size increases. Different spatial, temporal and social distributions affect the result as well.

Additionally, we evaluate the performance of SER by comparing that of TIPS and the results are in Table 4. For the compactness, we report how much more time SER averagely needs to find query answers by varying each factor of every parameter. The largest gap is when changing $\ell$ similar to the previous tests. Also, the efficiency of SER degrades as the size of data sets increases.
C. EXPERIMENT FOR AREA OF SEARCH SPACE

To study the effectiveness of pruning techniques, we compare the area of the search space. Note that the whole area is 10,000 because of the normalization of spatial space. Figure 12 shows the experimental results of the extent of search space with diverse data sets and parameters like previous experiments. Notably, we describe only NCF and TIPS for this experiment since the results of experiments on NCF and IPS are exactly equal in terms of these tests. Indeed, it is nothing worth comparing of to check the result of SER since it is a technique based on two data-driven structures. That is, the sum of two search spaces is extremely large due to overlaps of rectangles containing data points.

Figure 12(a) illustrates the effect of varying $\ell$ on BE. When $\ell$ is 1 or 2, the search spaces of both NCF and TIPS are quite small and similar because the number of pruned cells is still tiny. However, when $\ell$ exceeds 2, the gap between the results of two methods is much larger. Although the outcome of TIPS also changes, its ratio is far better because NCF is difficult to utilize time information for pruning search spaces, i.e., cells which cover $\ell$-hop friends in our study. In other words, the pruning technique of TIPS can be used well when the area of search space gets larger with the increase in the number of $\ell$-hop friends.

Next, the result of experiments on BE with varying $m$ shown in Figure 12(b). When $m$ is 2, the size of the search space of both methods is quite small because $m = 2$ means just to find the nearest $\ell$-hop friend of $u$. That is, the final friend might be around query locations. As $m$ increases, those of two methods grow because the algorithms continuously investigate spaces until finding $m - 1$ passengers.

Third, Figure 12(c) shows the results for varying $h$ on BE. We can observe that the area of search space is falling as $h$ increases. That is, larger $h$ gives doable performances. The gap between the results of the two methods gradually becomes narrow because our method finds candidate passengers and a result group more easily. Additionally, when $h$ is infinity, the outcomes of the two methods are similar because all passengers satisfying other constraints could be candidates in terms of the time constraint.

Finally, Figure 12(d) reports the results for varying $t_u$. In this experiment, a notable tendency is lots of space exploration at night. The result is highly related to the tests in Figures 11(d), (h), (l), and (p). Europe data sets are affected by driving timeslots than America data sets.

The same experiments on GE, BA, and GA are depicted in Figures 12(e)-(p). As we might expect, all trends of the experiments are similar to the tests on BE except in some cases. For example, we can see an interesting result of the test for varying $\ell$ on GA; the size of the search space of TIPS is getting smaller after $\ell$ is 3 distinct from the test on BE. This is because $\ell$-hop friends of $u$ can sufficiently be an optimal ridesharing group before checking more cells than that of the experiment when $\ell$ is 3. This outcome is remarkable when $\ell$ is 5.

D. EXPERIMENT FOR GROUP MATCHING RATE

We report group matching rate with diverse parameters in Figure 13. Note that the rate is changed by different data sets because our methods exactly find an answer if there is an optimal group. Since the default $m = 4$, we regard the returned group as a failure if the size of the group is under 4. Moreover, the results of all methods are the same, so we conducted these tests by varying data sets.

Figure 13(a) illustrates the experimental results for varying $\ell$. Intuitively, it is hard to group four (default $m = 4$) members in 1-hop friend list because all members have to be connected directly. Therefore, the rate is significantly worse when $\ell$ is 1. As $\ell$ increases, the rate of all methods grows as well. Interestingly, the general group matching rates of experiments on BE and GE are smaller than those of tests on BA and GA. The reason could be that passengers in America or Europe have different social graphs or spatial distribution regardless of the data size.

Next, Figure 13(b) describes the results for varying $m$. When $m$ is larger, the rate decreases due to the fixed number of $\ell$-hop friend list. That is, it is hard for all methods to find optimal groups using the limited number of candidates. Recall the experiments in Figures 11 and 12 for the correlation.

Then, Figure 13(c) displays the effect of varying $h$. Intuitively, $h$ is a huge constraint in terms of the matching rate. Thus, the result is surely affected by varying $h$ by filtering many passengers out. Furthermore, the data size and types also affect the group matching rate, so the result of the test on GE outperforms those of other experiments in contrast with previous tests. Another interesting thing is that the result on GE shows a quite high matching rate compared to that of BE even though two data sets were extracted from the same region. Thus the correlation between $h$ and GE quite high.
Lastly, the results of tests with varying \( t_u \) are depicted in Figure 13(d). The general outcome is similar to other experiments we discussed, but the matching rate at night on \( BE \) and \( GE \) is especially low. It is one of the reasons that friends of the driver might insufficiently stay nearby the driver at that time.

### E. EXPERIMENT FOR UPDATE TIME OF EXACT \( n \)-FRIEND SET

In our ridesharing system, the social update is more crucial than the spatial update because a simple grid update sufficiently can support the movement of objects. Therefore, we focus on the social update: insertion and deletion.

Figure 14 explains experiments for the update of the exact \( n \)-friend set. Since there are two types of updates, we separately conducted the tests and set \( n \) as 2 according to that we already discussed in Section V-A. When a friend \( v' \) of \( v \) is created, we have to check \( V_v^{(1)} \) for modifying \( F_v^{(2)} \). For the deletion of an edge \( e(v, v') \), \( V_v^{(1)} \) needs to be inspected again because the deletion affects \( F_v^{(2)} \). We compare our method in Section III-B, namely \( SCF \), to the basic method called \( Basic \), which updates the \( F_v^{(2)} \) by simply searching each \( V_v^{(1)} \).

As expected, \( SCF \) always outweighs \( Basic \), and the execution time increases as the size of the update or data set increases. Furthermore, the deletion of \( e(v, v') \) takes a longer time because we check whether every friend of the deleted vertex also has to be eliminated or not. If there is no edge between \( v \) and \( v' \), we get rid of the corresponding vertices.

### F. EXPERIMENT FOR VOLUMINOUS DATA SETS

Finally, we evaluate the performance of each method with varying data sizes. It looks pretty hard to gather instant passengers’ itinerary data over the size of data sets in the preceding experiments because the search space of every query is rebuilt after processing just before the query. Nonetheless, this simple experiment could be valuable to support lots of requests in a short time such as when a big concert or a crucial football match is over. For the experiments, we generated various sizes of synthetic data sets. Departure and arrival points are uniformly distributed, and social graphs are made on the basis of the Brábasi-Albert model [28]. The numbers of vertices are from 60 k to 300 k, and edges are from 300 k to 2.4 M.

Figure 15 explains execution time with varying data sizes on synthetic data sets. \( TIPS \) always achieves the best running time, and \( IPS \) is the following approach as expected. The gaps between each method become larger as the data size increases because especially the number of edges considerably affects the performance as we observed in Figures 11(a), (e), (i), and (m). Understandably, it is hard to compare that of \( SER \) due to its unusable performance.

### VI. RELATED WORK

We review related studies in this part. Research on ridesharing is introduced in Section VI-A, and geo-social studies are described in Section VI-B. Lastly, social-aware ridesharing is reviewed in Section VI-C.

#### A. RIDESHARING

The ridesharing has been a promising transportation system for not only reducing traffic issues but other related
problems. The work [1] divided a ridesharing problem into two views, called Join-based Ride Sharing and Search-based Ride Sharing. An approximated algorithm and a best-first approach win to tackle the problems with the enhancement of shared routes. Also, a taxi-sharing system, which deals with real-time requests, was developed in [3]. The system includes taxi-searching and scheduling algorithms, and especially considers vehicle capacity, time window, and monetary constraints. With regard to another system, [7] introduced Xhare-a-Ride targeting especially at in-memory indexing. The authors proposed and exploited clusters composed of landmarks for efficient ridesharing. Recently, [9] generalized the ridesharing to a new problem, Unified Route Planning for Shared Mobility. Especially, the proposed multi-objective function enables the problem to be feasible.

In addition to traditional concepts, the latest research on ridesharing has been studied with interesting objectives. In [35], a fair pricing model for drivers and riders was designed for increasing the revenue of the service provider. Its Auction-based Price-Aware Real-time ride-sharing framework satisfies the temporal and monetary constraints. The authors focused on a real-time environment as well. Another work [36] introduced Utility-aware Ridesharing on Road networks for maximizing the satisfaction of riders for worthwhile travel. The utility is composed of vehicles, riders, and trajectories in the work. Next, the study [37] proposed Activity-Based Ridesharing Algorithm to raise the rate of group matching by utilizing the activity information. Specifically, its active-based algorithm creates a pool of alternative destinations for effective ridesharing. The work has shown its effectiveness and efficiency by comparing trip-based approaches. Moreover, another study [38] tackled one of the ridesharing issues from the privacy point of view.

B. GEO-SOCIAL QUERIES

Social media has given researchers and developers opportunities to study geo-social queries in the past few years. In [39], a Circle of Friend Query searches the $k-1$ friends of a query user. The query was defined with the concept of diameter and the score of the ranking function. In another work [40], a geo-social query processing framework was proposed. The framework envelopes primitive queries, called Range Friends and Nearest Friends queries, which consider directed friends. Authors developed simple but effective algorithms for processing queries on centralized and distributed machines.

Other fundamental geo-social queries and those processing methods on Social Grid Index (SGI) were studied [32], [41]. Specifically, $\ell$-Close Range Friends queries retrieve all people who are connected in specified social boundaries of an input user within a spatial range. Similarly, $k$-Nearest $\ell$-Close Friend ($k\ell$-NCF) queries find $k$ nearest neighbors among $\ell$-hop friends of a query user. The authors developed several algorithms with an efficient update method for the SGI. Recall that one of the baselines is a group version of [32] because our social model, i.e., $k$-club, can be effectively decomposed by $k\ell$-NCF queries.

C. SOCIAL-AWARE RIDESHARING

With the development of ridesharing services, its relevant crime rate is getting higher. Although an increasing number of works have chosen a social-awareness to mitigate the problems, those are in its initial stage [17]–[21].

A work [17] searches socially aware groups for ridesharing services by proposing and processing Social-aware Ridesharing Group queries. Specifically, ridesharing member’s trip has to be similar to that of the driver, and a social-awareness is defined by $k$-core, which is one of the graph models. Every member of the group has at least $k$ direct friends among the members according to the definition of the model. Furthermore, the authors also suggested Social-Info R-tree, which accelerates their processing method.

Additionally, the research [18] has concentrated on the top-$k$ taxi searches in road networks. In the study, riders can select their favorable taxi among the top-$k$ taxis. Especially, the authors of the work proposed Top-$k$ Social-aware Taxi Ridesharing queries, which return the top-$k$ taxis meeting a trip request based on their ranking function.

A study [19] also has found ridesharing groups by defining Community aware Ridesharing Group queries. Contrary to other works, authors utilized not users’ friendship but the relationship at community levels because their objective includes avoiding privacy threats. That is, at least $m$ members in a formed group have to share at least $k$ communities. This concept is a modified $k$-core, so we put it in the same category.

Another work [20] has covered a problem called Assignment of Requests to Offers for the practical ridesharing in a similar environment to [18]. Authors proposed cost functions for evaluating a social distance and a common interest through an undirected social network and a modified Jaccard similarity. The problem is tackled by a linear programming method and heuristic techniques.
Recently, Price-aware Top-k Matching queries [21] were proposed on Hitch, one of the ridesharing modes. For comparing groups, the work uses widespread ranking functions, which consider the fare also, instead of cohesiveness models. Moreover, the authors suggested several algorithms with the Enhanced Safe-Region Quadtree.

Despite careful studies on ridesharing, their aims and methods are different from those in our study. For instance, the work [17] insufficiently studies a riding time and has a limited model to adjust the level of closeness because someone wants to also consider undirected friends, e.g., [49]. Additionally, the work [20] utilizes only driver-centered social comfort. It could make unsatisfactory groups for trustworthy ridesharing. Indeed, each problems assume different ridesharing modes [18], [20], [21]. Although some works [17], [19] used the same model, i.e., Slugging, constraints, and the social model are different from our work.

Meanwhile, we focus on efficiently finding cohesive ridesharing groups by considering empty seats, available timeslots, and itinerary. Besides, we adopt a cohesiveness
model for controlling an acceptable social bound. We summarize related works and compare those differences in Table 5.

VII. CONCLUSION AND FUTURE WORK

Nowadays, ridesharing services are being rapidly developed owing to the benefits in many fields. However, the cohesion-related problem of ridesharing groups also occurs. For cohesive ridesharing, we define a new problem, the ℓm-CRG query, taking into account the spatial, social, and temporal aspects. We also develop a series of algorithms for efficiently matching a driver and passengers in our framework. Furthermore, the efficiency of the proposed methods is proved using several data sets.

In the future, we will study an enhanced ridesharing system taking other aspects such as the shortest route and the interest of riders. Our method could be applied to various applications and the environment with the study on spatial networks, obstacles on the road, and location uncertainty. Furthermore, other ridesharing models can be an up-and-coming topic for future research.

APPENDIX

The SER is one of ℓm-CRG query processing methods for evaluating the performance. Its main concept is to encapsulate the R-tree and the social-awareness at once called the Social-Equipped R-tree (SER). For fair evaluation with others, the SER exploits the same concept of ℓ-hop friends as social processing, and a whole process is also similarly devised to the proposed method by considering the properties of [17].

The pseudo code of the main query processing is in Algorithm 5, and that of core functions is described in Algorithm 6.

ACKNOWLEDGMENT

This work is based on the dissertation of C. Shim conducted at Korea University [50].

REFERENCES

[1] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong, “An efficient ride-sharing framework for maximizing shared route,” IEEE Trans. Knowl. Data Eng., vol. 30, no. 2, pp. 219–233, Feb. 2018.
[2] M. Asghari and C. Shahabi, “ADAPT-pricing: A dynamic and predictive technique for pricing to maximize revenue in ridesharing platforms,” in Proc. 26th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst, Seattle, WA, USA: ACM, Nov. 2018, pp. 189–198.
[3] S. Ma, Y. Zheng, and O. Wolfson, “Real-time city-scale taxi ridesharing,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 7, pp. 1782–1795, Jul. 2015.
[4] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen, “Price-and-Time-Aware dynamic ridesharing,” in Proc. IEEE 34th Int. Conf. Data Eng. (ICDE), Paris, France, Apr. 2018, pp. 1061–1072.
[5] H. Qadir, O. Khalid, M. U. Khan, A. U. R. Khan, and R. Nawaz, “An optimal ride sharing recommendation framework for carpooling services,” IEEE Access, vol. 6, pp. 62296–62313, 2018.
[6] M. Asghari and C. Shahabi, “An on-line truthful and individually rational pricing mechanism for ride-sharing,” in Proc. 25th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. Redondo Beach, CA, USA: ACM, Nov. 2017, p. 7.
[7] R. S. Thangaraj, K. Mukherjee, G. Haravi, A. Metrewiet, N. Annamaleni, and K. Chattopadhyay, “Xshare-a ride: A search optimized dynamic ride sharing system with approximation guarantee,” in Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE), San Diego, CA, USA, Apr. 2017, pp. 1117–1128.
C. Shim et al.: Cohesive Ridesharing Group Queries in Geo-Social Networks

[34] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, “The anatomy of the facebook social graph,” 2011, arXiv:1111.4503. [Online]. Available: http://arxiv.org/abs/1111.4503

[35] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, “Price-aware real-time ride-sharing at scale: An auction-based approach,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, p. 3.

[36] P. Cheng, H. Xin, and L. Chen, “Utility-aware ridesharing on road networks,” in Proc. ACM Int. Conf. Manage. Data (SIGMOD). Chicago, IL, USA: ACM, 2017, pp. 1197–1210.

[37] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[38] CHANGBEOM SHIM received the B.S. degree in electronics and information engineering and the Ph.D. degree in computer science and engineering, from Korea University, Seoul, in 2014 and 2019, respectively. He is currently working as a Research Associate with the Department of Electrical and Computer Engineering, Curtin University, Australia. His research interests include spatial and graph algorithms and multi-object tracking.

[39] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, “Price-aware real-time ride-sharing at scale: An auction-based approach,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, p. 3.

[40] P. Cheng, H. Xin, and L. Chen, “Utility-aware ridesharing on road networks,” in Proc. ACM Int. Conf. Manage. Data (SIGMOD). Chicago, IL, USA: ACM, 2017, pp. 1197–1210.

[41] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[42] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[43] GYUHYEON SIM received the B.S. degree in computer science and engineering from Korea University, Seoul, in 2020. His research interests include spatial databases and machine learning.

[44] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[45] GYUHYEON SIM received the B.S. degree in computer science and engineering from Korea University, Seoul, in 2020. His research interests include spatial databases and machine learning.

[46] CHANGBEOM SHIM received the B.S. degree in electronics and information engineering and the Ph.D. degree in computer science and engineering, from Korea University, Seoul, in 2014 and 2019, respectively. He is currently working as a Research Associate with the Department of Electrical and Computer Engineering, Curtin University, Australia. His research interests include spatial and graph algorithms and multi-object tracking.

[47] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[48] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[49] Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[50] C. Shim, Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.

[51] C. Shim, Y. Wang, R. Kutadinata, and S. Winter, “Activity-based ridesharing: Increasing flexibility by time geography,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Burlingame, CA, USA: ACM, 2016, pp. 1–10.