Skyline Cohesive Group Queries in Large Road-social Networks

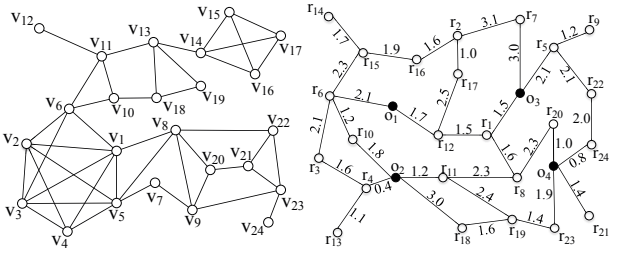
Qiyan Li*, Yuanyuan Zhu*, Jeffrey Xu Yu†

*School of Computer Science, Wuhan University, China
†The Chinese University of Hong Kong, Hong Kong, China
*{qiyan.li, yyzhu}@whu.edu.cn, †yu@se.cuhk.edu.hk

Abstract*—Given a network with social and spatial information, cohesive group queries aim at finding a group of users, which are strongly connected and closely co-located. Most existing studies limit to finding groups either with the strongest social ties under certain spatial constraint or minimum spatial distance under certain social constraints. It is difficult for users to decide which constraints they need to choose and how to decide the priority of the constraints to meet their real requirements since the social constraint and spatial constraint are different in nature. In this paper, we take a new approach to consider the constraints equally and study a skyline query. Specifically, given a road-social network consisting of a road network G_r and a location-based social network G_s, we aim to find a set of skyline cohesive groups, in which each group cannot be dominated by any other group in terms of social cohesiveness and spatial cohesiveness. We find a group of users using social cohesiveness based on (k, c)-core (a k-core of size c) and spatial cohesiveness based on travel cost to a meeting point from group members. Such skyline problem is NP-hard as we need to explore the combinations of c vertices to check whether it is a qualified (k, c)-core. In this paper, we first provide exact solutions by developing efficient pruning strategies to filter out a large number of combinations which cannot form a (k, c)-core, and then propose highly efficient greedy solutions based on a newly designed cd-tree to keep the distance on the road network and social structural information simultaneously. Experimental results show that our exact methods run faster than the brute-force methods by 2-4 orders of magnitude in general, and our cd-tree based greedy methods can significantly reduce the computation cost by 1-4 order of magnitude while the extra travel cost is less than 5% compared to the exact method on multiple real road-social networks.

I. Introduction

With the rising popularity of GPS-enabled mobile devices, users can easily access to location-based service (LBS) such as Google Maps, Foursquare, Facebook Places, Meetup, etc. The demand for LBS boosts the emergence of location-based social networks, which can bridge the gap between the physical world and the virtual world of social networks and enable the organization of group-based impromptu activities. For example, a party organizer may need to choose a place to hold the party and decide whom to invite; a basketball player who wants to have an impromptu 3v3 game may need to find a nearby playground and those he/she can play with; a player of a live AR game such as Pokemon Go needs to check where and with whom he/she can play the game. All above examples require the social and spatial cohesiveness of groups, i.e., the selected group members need to have a good social relationship to create a pleasant atmosphere in the activity and have small distances to a rallying point to save the travel cost, which are essentially cohesive group queries on geo-social networks.



(a) Social Network G_s (b) Road Network G_r

Fig. 1. A Motivation Example

Motivation. Cohesive group queries on geo-social networks have been attracting increasing research efforts in recent years [1]–[9] . In these works, the spatial cohesiveness is measured by the spatial distance, and the social cohesiveness is measured by k-core [4]–[8], k-truss [9], k-plex [1], and other density measures [2], [3]. Besides, [10] studies the problem of finding k-core groups with high similarities between group members, which can be adapted to deal with geo-social group queries if the spatial distance is converted to similarities. Although these works consider both geo and social information, there are still gaps between them and real-life applications, and particularly they have the following major limitations:

1) *Lack of flexibility.* To conduct cohesive group queries on geo-social networks, we need to consider both social cohesiveness and spatial cohesiveness. However, most existing studies mainly aim at finding maximum/minimum groups under certain social and spatial cohesiveness constraints [4] [6] [10] [9] [5], or finding the groups with the maximum social cohesiveness under specific spatial cohesiveness constraints [8]. Based on the model of densest k-cores, [8] provides a heuristic method to select the group with the smallest travel cost, but they lack the flexibility to balance the social cohesiveness and the spatial cohesiveness. [7] tried to define a score to combine social and spatial cohesiveness, but their balancing parameters are hard to be specified, as the social cohesiveness and spatial cohesiveness are usually unknown in previous.

2) *Impracticality of spatial distance.* Most existing works [4]–[7], [9], [10] measure the spatial distance based on euclidean space. However, in real-world applications, the distance along road-network is a more accurate way to model the

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00041

travel cost. To our best knowledge, [8] is the first work that considers cohesive group queries based on the road network. However, they do not consider the balance between the social cohesiveness and the spatial cohesiveness and only provide a heuristic solution.

We use the following example to illustrate the above limitations that motivate our study.

Example 1.1. Consider a road-social network in Figure 1, where vertices and edges represent users and their social connections in Figure 1(a) and represent road junctions and road segments in Figure 1(b). The location of user v_i in Figure 1(a) is the vertex r_i in the road network in Figure 1(b). The black points in Figure 1(b) are points of interest (POI) which can be considered as the candidate meeting points. Suppose the querying vertex is v_1 in Figure 1. We compare two cohesive groups, one is C_1 induced by $\{v_1, v_2, v_3, v_4, v_5\}$ and the other is C_2 induced by $\{v_1, v_5, v_7, v_8, v_9\}$. C_1 is a 4-core in social network and o_1 is the optimal meeting points for C_1 with total travel cost 24.9. C_2 is a 2-core, which is not as cohesive as C_1, but its travel cost to o_3 is only 13.0. A flexible query should be able to return both C_1 and C_2 for the querying user so that the user can make a choice based on whether he/she prefers a stronger connected group or a smaller travel cost.

In this paper, we argue that the social cohesiveness and spatial cohesiveness are two inherently different measures, and cohesive group queries can be naturally modeled as a skyline query. Thus for the first time, we study the problem of finding a set of skyline cohesive groups over road-social networks. Let $\mathcal{G} = (G_r, G_s)$ be a road-social network where G_r is the road network and G_s is the location-based social network and O be a set of candidate meeting points. For a given query $Q = (u_q, c, r)$ where u_q is the querying user, c is the group size, and r is the distance threshold, we aim to find a set of skyline cohesive groups, $\mathcal{C}$, that contain u_q and satisfy the spatial and size constraints such that $\forall C_1 \in \mathcal{C}$, C_1 cannot be dominated by any other group in terms of social cohesiveness and spatial cohesiveness. Here a group C_1 dominates another group C_2 if and only if it is better than C_2 in one dimension and no worse than C_2 in another dimension. We model the social cohesiveness by (k, c)-core, which is a k-core of size c, and evaluate the spatial distance by travel cost, which is the sum of the shortest path distance from users to the optimal meeting point.

Challenges and Contributions. The skyline cohesive group query problem is NP-hard as we need to explore a large number of combinations of c vertices to check whether it is a qualified (k, c)-core. However, (k, c)-core enumeration problem over road-social networks has not been well studied in the literature due to its NP-hardness. Although [8] studied the problem of finding the densest groups with minimum travel costs, as the special case of skyline cohesive group query, they do not directly solve the (k, c)-core enumeration problem in the road-social network. Instead, they proposed a heuristic solution by first finding a maximal k-core containing u_q with more than c users with the largest k value and then choosing c users from the maximal k-core as the groups. They may miss the densest (k, c)-core, because choosing c users in a maximal k-core may only form a $(k-2)$-core, but choosing c users in a $(k-1)$-core may still lead to a $(k-1)$-core. To tackle this problem, we first provide an efficient exact solution by exploiting structural properties of (k, c)-cores to filter out a large number of vertex combinations that cannot lead to (k, c)-cores. Then we propose a highly efficient greedy solution based on a newly designed cd-tree that can keep the distance and social information simultaneously for the efficient computation of (k, c)-core. The main contributions of this work are summarized as follows.

- We define a new skyline cohesive group model, which can measure the cohesiveness of both social and spatial dimensions flexibly. Based on this model, we formulate a family of cohesive group query problems over road-social networks to capture different variants of application scenarios in the real world.
- We propose exact algorithms with effective pruning techniques that can deal with large graphs with million edges.
- We propose efficient greedy solutions based on a newly designed cd-tree to keep the distance and social information simultaneously, which can significantly accelerate the computation without too much loss of result quality.
- We conducted extensive experiments on real datasets to show the effectiveness and efficiency of the exact algorithms and the cd-tree based greedy algorithms.

Roadmap. We define our problems in Section II. Section III presents the exact solutions. Section IV gives the cd-tree index and Section V presents the greedy algorithms. Experimental studies are reported in Section VI. We review the related work in Section VII and conclude the paper in Section VIII.

II. Problem Statement

In this paper, we consider the road-social network as a composition of a pair of networks, a road network G_r, and a social network G_s, denoted as $\mathcal{G} = (G_r, G_s)$. They are both simple undirected graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$, where vertices and edges represent users and their relationships respectively in the social network G_s, and represent road junctions and segments respectively in the road network G_r. Each vertex $v \in V$ is associated with a location $(v.x, v.y)$, where $v.x$ and $v.y$ denote its spatial positions along the x-axis and y-axis in a two-dimensional space. We use $V(G)$ and $E(G)$ to denote the vertex set and the edge set, and use $|V(G)|$ and $|E(G)|$ to denote the number of vertices and number of edges in G respectively. For a vertex $v \in V(G)$, we denote the set of its neighboring vertices by $N_G(v) = \{u \in V | (u, v) \in E\}$ and its degree by $\delta_G(v) = |N_G(v)|$. For a path $s \rightsquigarrow t$ between vertices $s, t \in V(G)$ in graph G, its length is the sum of weights of edges along the path. The distance between s and t is the length of the shortest path, denoted as $dist_G(s, t)$. In the following, we simplify the above notations as V, E, $N(v)$, $\delta(v)$, $dist(s, t)$ if the context is clear.

Definition 2.1 (k-core). Given a graph $G = (V, E)$ and an integer k, a k-core is a connected subgraph $H \subseteq G$, such that $\forall v \in V(H)$, $\delta_H(v) \geq k$.

The core number of a subgraph $H \subseteq G$ is the minimum degree of the vertices in H, defined as $core(H) = \min_{v \in V(H)} \delta_H(v)$. The core number of $v \in V(G)$ is the maximum core number of subgraphs containing v, i.e., $core_G(v) = \max_{H \subseteq G \wedge v \in V(H)} core(H)$. A k-core $H \subseteq G$ is called a *maximal k-core*, denoted as k-$\widehat{core}$, if there exists no supergraph H' of H ($H \subseteq H' \subseteq G$) with the same core number k.

Definition 2.2 ((k, c)-core). Given a graph G and an integer k, a (k, c)-core $H \subseteq G$ is a k-*core* that has exactly c vertices.

Definition 2.3 (Road-social group). Given a road-social network $\mathcal{G} = (G_r, G_s)$, a set of candidate meeting points O, a querying user u_q, a distance threshold r, integers k and c, a road-social group is a subgraph $C \subseteq G_s$ such that:
1) C is a (k, c)-core containing u_q;
2) $\exists o \in O$ such that $dist(o, V(C)) \leq r$ where $dist(o, V(C)) = \max_{v \in V(C)} dist(o, v)$.

Definition 2.4 (Skyline Road-social Groups). Given a road-social network $\mathcal{G} = (G_r, G_s)$ and a set of candidate objects O, $\mathcal{C}$ is a set of skyline road-social groups if $\forall C_1 \in \mathcal{C}$ cannot be dominated by any other group $C_2 \in \mathcal{C}$. Here, C_2 dominates C_1 if and only if:
1) $core(C_2) > core(C_1)$ and there exist objects o_1 and o_2 such that $cost(o_2, V(C_2)) \leq cost(o_1, V(C_1))$ where $cost(o, V(C)) = \sum_{v \in V(C)} dist(o, v)$, or
2) $core(C_2) = core(C_1)$ and there exist objects o_1 and o_2 such that $cost(o_2, V(C_2)) < cost(o_1, V(C_1))$.

Road-social group query (RSGQ) Problem. Consider a road-social network $\mathcal{G} = (G_r, G_s)$ and a set of objects O, RSGQ aims to find road-social groups $\mathcal{C}$ under certain social and spatial requirements. Below, we list three problems. $\mathsf{RSGQ_S}$ is the last where the first two are the special cases of $\mathsf{RSGQ_S}$.

1) Densest road-social group search ($\mathsf{RSGQ_D}$): Given a query $Q = (u_q, c, r)$ where u_q is the querying user, c is the group size and r is the distance threshold, find a group C and an object $o \in O$ such that $core(C)$ is maximized and $cost(o, V(C))$ is minimized among all the groups with the same core number.
2) Location-specified skyline road-social group search ($\mathsf{RSGQ_{LS}}$): Given a query $Q = (u_q, c, r, k_{min}, o)$ where k_{min} is the minimum core number and $o \in O$ is the given meeting point, find a set of skyline road-social groups $\mathcal{C}$ such that $\forall C \in \mathcal{C}$, $core(C) \geq k_{min}$.
3) Skyline road-social group search ($\mathsf{RSGQ_S}$): Given a query $Q = (u_q, c, r, k_{min})$, find a set of skyline road-social groups $\mathcal{C}$ such that $\forall C \in \mathcal{C}$, $core(C) \geq k_{min}$.

Here, $\mathsf{RSGQ_D}$ and $\mathsf{RSGQ_{LS}}$ are special cases of $\mathsf{RSGQ_S}$ with certain social constraints and geo constraints, respectively. Specifically, if a user cares more about the social connectivity and has no strong preference for meeting points, he/she can perform the $\mathsf{RSGQ_D}$ search; if a user has set the meeting point o and does not have a strong constraint on social connectivity, he/she can perform the $\mathsf{RSGQ_{LS}}$ search. We design pruning techniques for these two different problems, which can finally pave the way to solve the $\mathsf{RSGQ_S}$ problem.

Example 2.1. Suppose v_1 is user Alice at location r_1 in Figure 1. She wants to play a board game of 5 people in one of the four nearby clubs at o_1, o_2, o_3 and o_4. To make it convenient for each player, the distances between the selected club and all 5 people should be less than 7km. She would like to conduct RSGQ queries to get the recommendation of suitable players and clubs. 1) If she wants to play with a group of close friends, she can perform the $\mathsf{RSGQ_D}$ query, and the result is $(\{v_1, v_2, v_3, v_4, v_5\}, o_1, 4, 24.9)$. In this result, the participants are v_1, v_2, v_3, v_4, v_5, the core number of the induced subgraph is 4, the meeting point is o_1 and the travel cost is 24.9. 2) If she has already decided to go to the club at o_1, providing a minimum familiarity constraint k = 2, she can conduct the $\mathsf{RSGQ_{LS}}$ query, which returns $(\{v_1, v_2, v_3, v_4, v_5\},\ o_1,\ 4,\ 24.9)$, $(\{v_1, v_2, v_3, v_4, v_6\},\ o_1,\ 3,\ 20.2)$ and $(\{v_1, v_3, v_6, v_{10}, v_{11}\},\ o_1,\ 2,\ 19.1)$. 3) If she has not decided where to go and also has no strong social preference, she can perform the $\mathsf{RSGQ_S}$ query, which returns $(\{v_1, v_2, v_3, v_4, v_5\},\ o_1,\ 4,\ 24.9)$; $(\{v_1, v_2, v_3, v_4, v_6\},\ o_1,\ 3,\ 20.2)$ and $(\{v_1, v_5, v_7, v_8, v_9\},\ o_3,\ 2,\ 13.0)$.

Problem Hardness Analysis. First, we analyze the hardness of finding (k, c)-cores containing a query u_q, which are involved by all three cases in the RSGQ problem. It's well known that finding a k-$\widehat{core}$ can be easily completed in $O(m)$ time [11] where m is the number of edges in the graph. However, finding a k-core consisting of exactly c vertices is much more difficult. The decision version of (k, c)-core search problem is as follows.

(k, c)-core Problem. Given a graph G, a vertex u_q, integers k and $c \geq k$, test whether there is a connected k-core with c vertices containing u_q.

Theorem 2.1. *(k, c)-core problem is NP-complete.*

Proof. We prove this by reducing the well-known maximum clique decision problem to (k, c)-core problem. Given a graph G and an integer k, the maximum clique decision problem is to check whether G contains a clique of size k. Now we construct an instance of (k, c)-core problem, consisting of a graph G, parameters k and $c = k + 1$. Next, we show that the instance of the maximum clique decision problem is a YES-instance iff the instance of (k, c)-core problem is a YES-instance. Clearly, any clique with k vertices is a connected k-core with size $c = k + 1$. On the other hand, given a solution H for (k, c)-core problem, H must contain k vertices since H is a k-core and $c = k + 1$, which implies H is a clique. □

Corollary 2.1. *All the three* RSGQ *problems are NP-hard.*

This corollary can be easily derived, as all three cases of the RSGQ problem involve the operation of finding (k, c)-cores containing a query u_q, which is an optimization version of the decision version of (k, c)-core problem.

III. Exact Solutions

In this section, we give the exact solutions to solve RSGQ problems. From the definition of (k, c)-core, we know that a (k, c)-core can only be found in a k-$\widehat{core}$ with at least c vertices. Thus, a natural way to find (k, c)-cores is checking whether each combination of c vertices in the k-$\widehat{core}$ can eventually form a k-core. Based on such intuition, we can derive the following brute-force methods.

Brute-force methods. (1) $\mathsf{RSGQ_D}$. To solve this problem, we first find the maximum core number k_{max} such that (k_{max}, c)-cores exist and then check all possible (k_{max}, c)-cores to find a group and an object with the minimum travel cost. In the beginning, to reduce the search space, we conduct the range query to filter out all the objects and users that do not satisfy the distance threshold r and obtain the induced subgraph G'_s by the remaining users in the social network. The range query can be accelerated by G-tree [12] or G*-tree [13]. Then, we do core decomposition [11] on G'_s. Starting from the largest possible core value $k = \min\{core(u_q), c-1\}$, we extract k-$\widehat{core}$ containing u_q, $G_s'^{[k]}$, from G'_s, and enumerate all the possible connected subgraphs induced by c vertices containing u_q to check whether it is a (k, c)-core. If yes, we will record this group and the corresponding object with the minimum travel cost. If no (k, c)-core can be found after the enumeration, we will decrease k by one and repeat the above process until (k, c)-cores are found. At last, the group with the minimum travel cost is returned. (2) $\mathsf{RSGQ_{LS}}$. Similar to $\mathsf{RSGQ_D}$, we first filter users by range query and do core decomposition. Starting from the largest possible core value $k_{max} = \min\{core(u_q), c-1\}$, we gradually enumerate all the (k, c)-cores to find a group with the minimum travel cost to the given meeting point o. Such a process is repeated until $k < k_{min}$, and the set of skyline road-social groups will be finally returned. (3) $\mathsf{RSGQ_S}$. Similarly, we will first perform the range query to obtain G'_s. Then for k from $\min\{core(u_q), c-1\}$ to k_{min}, we find the (k, c)-core and the corresponding object with the minimum travel cost and finally return the skyline cohesive groups.

However, the above brute-force methods are quite time-consuming, as we need to enumerate the (k, c)-cores $C(|V(G_s'^{[k]})|, c)$ times in the worst case, which is impractical for large graphs as k-$\widehat{core}$ $G_s'^{[k]}$ is usually also very large. In fact, many combinations of c vertices actually cannot lead to a qualified (k, c)-core. Thus, in the following, we will exploit the properties of (k, c)-core, and propose more efficient algorithms to solve RSGQ problems accordingly.

A. Exact $\mathsf{RSGQ_D}$ Algorithm

To solve this problem, [8] has provided a heuristic solution by first finding k-$\widehat{core}$ containing u_q with more than c users with the maximum k value and then choosing c users from the k-$\widehat{core}$ as the group. However, such a heuristic method may not return the densest group due to the following reasons: 1) k may not be maximum. For example, choosing c users in a k-$\widehat{core}$ may only form a $(k-2)$-core, but choosing c users in a $(k-1)$-core may still lead to a $(k-1)$-core. 2) Even if a (k, c)-core C with the maximum k is found, there may not exist a meeting point o such that $dist(o, V(C)) \leq r$. 3) Even if k is maximum and the meeting point is found, there may be other groups that are spatially closer. For example, we may find $(k-1)$-cores of c users with the smallest travel cost in a k-$\widehat{core}$, but we may also find $(k-1)$-core of c users in a $(k-1)$-$\widehat{core}$ with even smaller travel cost.

In this paper, to solve the $\mathsf{RSGQ_D}$ accurately and efficiently, we will first exploit the properties of n-plex, which is closely related to (k, c)-core.

Definition 3.1 (n-plex) Given a graph G and an integer n, an n-plex is a connected subgraph $H \subseteq G$, such that $\forall v \in V(H)$, $\delta_H(v) \geq |V(H)| - n$. H is a maximal n-plex if there is no any other n-plex $H' \supseteq H$, denoted as n-$\widehat{plex}$ [14].

The above definition shows that each vertex should connect with at least $|V(H)| - n$ vertices, leading to the following equivalence property between cores and plexes.

Property 3.1. (Equivalence property) A (k, c)-core is a $(c-k)$-plex of size c.

Next, we will first show the closed property of n-plex, and then build the connections among (k, c)-core, $(c-k)$-$\widehat{plex}$ and k-$\widehat{core}$.

Property 3.2. (Closed property of n-plexes) n-plexes are closed under exclusion, i.e., given an n-plex H, any connected subgraph $H' \subseteq H$ is still an n-plex [14].

Theorem 3.1. *A (k, c)-core is contained in a $(c-k)$-$\widehat{plex}$, and any $(c-k)$-$\widehat{plex}$ must be contained in a k-$\widehat{core}$ with at least c vertices.*

Proof. Let H be a (k, c)-core, which is also a $(c-k)$-plex of size c (Property 3.1). From Definition 3.1, we can derive that H must be contained in a $(c-k)$-$\widehat{plex}$ H' with $c' \geq c$ vertices, and for each vertex $v \in V(H')$, $\delta_v \geq c' - (c-k) \geq k$. Therefore, H' is also a k-core (Definition 2.1) and must be contained in a k-$\widehat{core}$ with at least c vertices. □

For example, the subgraph induced by $\{v_8, v_{20}, v_{21}, v_{22}, v_{23}\}$ is a (2,5)-core, which is also a 3-plex contained in the 3-$\widehat{plex}$ induced by $\{v_8, v_9, v_{20}, v_{21}, v_{22}, v_{23}\}$. This 3-$\widehat{plex}$ is contained in a 2-$\widehat{core}$ induced by all the vertices except $\{v_{12}, v_{24}\}$.

Based on Lemma 3.1 and Property 3.2, To find (k, c)-cores, we do not need to check all combinations in k-$\widehat{core}$ to get the candidate group. Instead we only need to find the $(c-k)$-$\widehat{plex}$ containing the query vertex u_q, and then any connected subgraph consisting of u_q and other $c-1$ vertices in the $(c-k)$-$\widehat{plex}$ is a (k, c)-core. However, the querying vertex can be contained in many maximal plexes, and finding all $(c-k)$-$\widehat{plex}$es in a large graph by the current enumeration framework is usually time-consuming [15]. To further reduce the search space, we use the bounded diameter property of $(c-k)$-plex [16] to prune unqualified vertices.

Property 3.3. (Bounded diameter of $(c-k)$-plex) Let H be a $(c-k)$-plex of size c and $diam(H)$ be the diameter of H.

Algorithm 1: RSGQ$_{\mathsf{D}}$-exact

Input : $\mathcal{G} = (G_r, G_s)$, O, u_q, c, r
Output: Densest road-social groups $\mathcal{C}$

1 $O' \leftarrow$ find objects by RangeQuery(u_q, r);
2 $G'_s \leftarrow$ find users by RangeQuery(o, r), $\forall o \in O'$;
3 do core decomposition on G'_s ;
4 $k \leftarrow \min\{core(u_q), c-1\}$;
5 **while** $k > 1$ **do**
6 $\quad G_s'^{[k]} \leftarrow$ extract k-$\widehat{core}$ containing u_q from G'_s;
7 $\quad G_s''^{[k]} \leftarrow$ DiameterPrune$(G_s'^{[k]})$;
8 $\quad \mathcal{C} \leftarrow$ ExactKCCore$(c, k, \{u_q\}, V(G_s''^{[k]})\backslash\{u_q\}, O)$;
9 $\quad$ **if** $\mathcal{C} \neq \emptyset$ **then return** $\mathcal{C}$;
10 $\quad k \leftarrow k-1$;
11 **Procedure** ExactKCCore (c, k, V_S, V_T, O)
12 **if** $|V_S| = c$ **then**
13 $\quad$ update $\mathcal{C}$ based on V_S and O;
14 **for** *each* $v \in V_T$ **do**
15 $\quad V'_S \leftarrow V_S \cup \{v\}$; $V_T \leftarrow V_T \setminus \{v\}$;
16 $\quad V'_T \leftarrow \{v' \in V_T | V'_S \cup \{v'\}$ is a $(c-k)$-plex and $N(v') \cap N(v) \geq 2k-c+2\}$;
17 $\quad O' \leftarrow$ common meeting points of V'_S;
18 $\quad$ ExactKCCore (c, k, V'_S, V'_T, O');

1) If $c < 2k+2$, then $diam(H) \leq 2$;
2) If $c \geq 2k+2$, then $diam(H) \leq c-2k+2$.

The RSGQ$_{\mathsf{D}}$**-exact Algorithm.** Algorithm 1 gives the general process of finding exact results for the RSGQ$_{\mathsf{D}}$ problem. We first filter out the objects and users that do not satisfy the distance constraint r by the range query function RangeQuery (lines 1-2), which can be accelerated by G-tree [12] or G*-tree [13]. We also further prune objects with less than c users inside the range r and obtain the whole set of candidate users by uniting all the candidate users for the remaining objects. Next, we do core decomposition [11] on the filtered social subgraph G'_s to compute the core number of each vertex (line 3), and initialize k as the largest possible value, $\min\{core(u_q), c-1\}$ (line 4). Then we gradually check k in a decreasing manner until we find (k,c)-cores with the minimum travel cost in lines 5-10. Before we enumerate the (k,c)-cores, we first prune vertex from G'_s that cannot form a k-$\widehat{core}$ based on Theorem 3.1 (line 6) or does not satisfy the diameter constraint based on Property 3.3 (line 7). The (k,c)-cores are enumerated by Procedure ExactKCCore (line 8) given below.

Procedure ExactKCCore**.** The process of finding (k,c)-cores is inspired by the plex enumeration framework [15], which returns all the $(c-k)$-$\widehat{plex}$es containing u_q by the branch-and-bound search. Here, we modify this framework to only find $(c-k)$-plexes of size c by terminating a branch as soon as the size of the $(c-k)$-plex reaches c to save searching time. Generally speaking, we keep a set of users V_S that already is a $(c-k)$-plex and expand this set by adding candidate users from set V_T until its size reaches c. Here we use set V_T to keep the set of candidate users that may form a (k,c)-core with users in V_S. To further prune unpromising vertices during the enumeration, we further exploit the following property of $(c-k)$-plex.

Algorithm 2: RSGQ$_{\mathsf{LS}}$-exact

Input : $\mathcal{G} = (G_r, G_s)$, o, u_q, c, r, k_{min}
Output: Skyline road-social groups $\mathcal{C}$

1 $G'_s \leftarrow$ find users in G_s by RangeQuery(o, r);
2 $k \leftarrow k_{min}$;
3 $G_s'^{[k]} \leftarrow$ extract k-$\widehat{core}$ containing u_q from G'_s;
4 $G_s''^{[k]} \leftarrow$ DiameterPrune$(G_s'^{[k]})$;
5 $\mathcal{C} \leftarrow$ ExactKCCore$(c, k, \{u_q\}, V(G_s''^{[k]})\backslash\{u_q\}, \{o\})$
6 **return** $\mathcal{C}$;

Property 3.4. (Least common neighbor of $(c-k)$-plex) Let H be $(c-k)$-plex of size c. Any two vertices $u, v \in V$ will have at least $2k-c+2$ common neighbors in H [15][1].

Based on the above property, users that do not share enough neighbors can be discarded in line 16. Since we have already computed all the distances between candidate users and meeting points in RangeQuery, we can get candidate groups based on the users and their common meeting points and choose the group with the smallest distance in line 13.

The complexity of k-core decomposition is $O(|E(G_s)|)$. Let F be the time complexity of RangeQuery supported by G-tree or G*-tree. The complexity of getting k-$\widehat{core}$ containing u_q is $O(|E(G'_s)|)$ and the complexity of DiameterPrune is $O(|E(G_s'^{[k]})|)$. Thus the overall time complexity of Algorithm 1 is $O(|E(G_s)|+F+\sum_{k=1}^{k_{max}} C(|V(G_s''^{[k]})|, c))$, where $k_{max} = \min\{core(u_q), c-1\}$ and $C(|V(G_s''^{[k]})|, c)$ is the worst-case time complexity of ExactKCCore. In fact, ExactKCCore is much faster in practice than the enumeration in the brute-force method as many branches can be pruned based on the above properties of the $(c-k)$-plex.

B. Exact RSGQ$_{\mathsf{LS}}$ *Algorithm*

For the RSGQ$_{\mathsf{LS}}$ problem, we have to find a set of skyline groups, i.e., all the (k,c)-cores need to be checked to find a (k,c)-core with the minimum travel cost for all possible $k \geq k_{min}$. Fortunately, we can avoid enumerating (k,c)-cores for each k value by exploring the connection between (k,c)-cores with different core numbers.

Property 3.5. (Nested property of k-core) k-cores are nested, i.e., any k'-core with $k' \geq k$ must be contained in a k-$\widehat{core}$.

Property 3.6. (Nested property of n-plex) n-plexes are nested, i.e., any n'-plex with $n' \geq n$ must be contained in an n-$\widehat{plex}$ [14].

Based on the above properties, together with the closed property of n-plex, we have the following theorem.

[1]Here a vertex v is also considered as the neighbor of itself.

Theorem 3.2. *Given a minimum core number k_{min} and a group size c, a (k,c)-core with $k \geq k_{min}$ must be contained in a $(c-k_{min})$-$\widehat{plex}$ with more than c vertices.*

Proof. Since plexes are nested, the $(c-k_{min}-1)$-$\widehat{plex}$, $(c-k_{min}-2)$-$\widehat{plex}$, $(c-k_{min}-3)$-$\widehat{plex}$, ... with more than c users are also contained in a $(c-k_{min})$-$\widehat{plex}$. Then the connected subgraphs consisting of c users in the $(c-k_{min}-1)$-plexes, $(c-k_{min}-2)$-plexes, $(c-k_{min}-3)$-plexes, ... are actually $(k_{min}+1,c)$-cores, $(k_{min}+2,c)$-cores, $(k_{min}+3,c)$-cores,.... Therefore, we can get all the (k,c)-cores for each possible core number $k \geq k_{min}$ by extracting subgraphs of c vertices in $(c-k_{min})$-$\widehat{plex}$. □

Based on Theorem 3.2, we give the process of finding skyline road-social groups for the RSGQ$_{\mathsf{LS}}$ problem in Algorithm 2, which shares a similar ExactKCCore function as in Algorithm 1, but differs from Algorithm 1 in the following aspects. First, since there is only one object, we only filter the users by RangeQuery in line 1 only by the given object o, and the set of objects O in ExactKCCore function is replaced with $\{o\}$ in line 5. Second, we fix k to be k_{min} (line 1) instead of gradually decreasing k from the largest core value (line 2). Third, Algorithm 2 returns all the skyline road-social groups while Algorithm 1 only returns the densest group.

Similarly, since we only do ExactKCCore once, we can derive that the overall complexity of Algorithm 2 is $O(|E(G_s)|+F+C(|V(G_s'^{[k_{min}]})|,c))$.

C. *Exact* RSGQ$_{\mathsf{S}}$ *Algorithm*

For the RSGQ$_{\mathsf{S}}$ problem, an intuitive solution is to perform RSGQ$_{\mathsf{LS}}$ for each candidate object and maintain a set of skyline road-social groups. However, it is impractical as even one RSGQ$_{\mathsf{LS}}$ search may cost a lot of time. A more efficient method is to consider the union of all users first and then find a meeting point, just like in RSGQ$_{\mathsf{D}}$. First, we get the sets of candidate meeting points and candidate users RangeQuery as well as the subgraph induced by the candidate users G_s' containing u_q (like lines 1-2 in Algorithm 1). Then, we further extract the k_{min}-$\widehat{core}$ and prune it by DiameterPrune (like lines 2-4 in Algorithm 2). Finally, we use ExactKCCore to get all the groups to meet the requirement of skyline groups (like line 5 in Algorithm 2 with o replaced by O').

Similar to Algorithm 2, the overall time complexity of the above solution is $O(|E(G_s)|+F+C(|V(G_s'^{[k_{min}]})|,c))$.

IV. Core-distance Tree

As mentioned before, the main challenge of the RSGQ problems is the enumeration of (k,c)-cores. It leads us to the following question: do we have to check all the (k,c)-cores to get a (k,c)-core with the smallest cost for a specific k? A possible solution is: for each candidate objects, we greedily select the c closest users in a k-$\widehat{core}$ and then check whether they can form a (k,c)-core. Then, among all the candidate objects, we return the object and the (k,c)-core with the smallest travel cost. By greedy search, we can avoid enumerating all the (k,c)-cores in k-$\widehat{core}$. However, such a greedy search method may still be time-consuming because the c closest users may not be a (k,c)-core. Thus, in this section, partially inspired by the cl-tree [17] that organizes an attributed graph, we first propose a new structure cd-tree to keep the core and distance information, and then based on this structure we give an efficient greedy search strategy to find (k,c)-cores with minimum travel cost approximately.

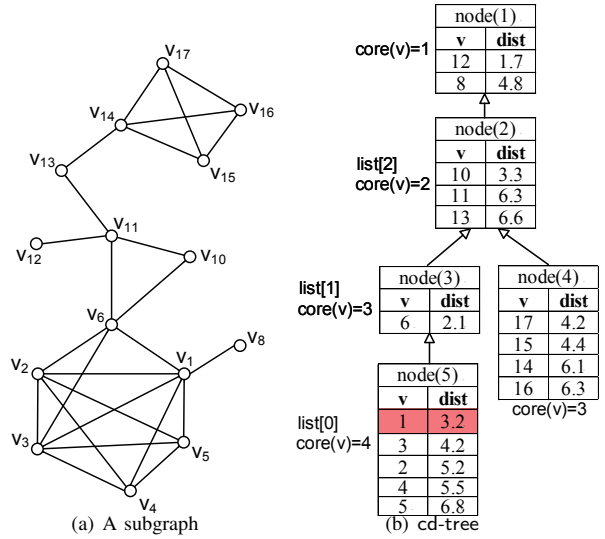


Fig. 2. An example of cd-tree

A. cd-tree *Structure*

First, we introduce a basic tree structure to organize the vertices of a graph based on the core numbers and the connections of different k-cores. To build such a tree, we first do core decomposition on this graph to obtain the core number of each vertex and k-$\widehat{core}$ for each $1<k<k_{max}$ where k_{max} is the largest core value in the graph. Then, we store each k-$\widehat{core}$ in a tree node. If a k'-$\widehat{core}$ is contained in a k-$\widehat{core}$, the node for the k'-$\widehat{core}$ becomes the child of the node for the k-$\widehat{core}$. Since the vertex set in a node is the superset of the vertex set of its child nodes, we can remove redundant nodes shared by its children to save space, and thus each vertex is only stored once in the tree.

Next, we consider how to store the distance information in this tree. For a meeting point o, we can extract the subgraph under a certain distance threshold r and build the tree structure to keep the core number. Then, for each node in the tree, besides the parent node, child nodes, core number, and the corresponding set of vertices, we also store distances between the vertices in this node and the meeting point and sort the vertices in ascending order of distance. Such a structure is called the cd-tree, as illustrated by the following example.

Example 4.1. First, we call RangeQuery $(o_1,7.0)$ to get the induced subgraph for object o_1 in Figure 1, as shown in Figure 2(a). We then group users based on core numbers and the connections of different k-cores and build the cd-tree, as shown in Figure2(b). Here, vertices in $node(5)$ form a 4-$\widehat{core}$, and vertices in $node(3)$ and $node(4)$ represent two

Algorithm 3: GreedyKCCore (c, k, V_S, V_T)

```
1  if |V_S| = c then
2      return the subgraph induced by V_S;
3  for each v ∈ V_T do
4      V'_S ← V_S ∪ {v}; V_T ← V_T \ {v};
5      V'_T ← {v' ∈ V_T | V'_S ∪ {v'} is a (c − k)-plex
         and N(v') ∩ N(v) ≥ 2k − c + 2};
6      G_k ← extract k-core̅ from G[V'_S ∪ V'_T];
7      if |V(G_k)| ≥ c and V'_S ⊆ V(G_k) then
8          V'_T ← V(G_k) \ V'_S;
9          C ← GreedyKCCore(c, k, V'_S, V'_T, o);
10         if C ≠ ∅ then return C;
11     G'_k ← extract k-core̅ of G[V_S ∪ V_T];
12     if V(G'_k) ≥ c and V_S ⊆ V(G'_k) then
13         V_T = |V(G'_k)| − V_S;
14     else break;
15 return ∅;
```

node sets with the same core number 3 but belong to two different 3-$\widehat{core}$s. Since the 4-$\widehat{core}$ in $node(5)$ is contained in the 3-$\widehat{core}$ consisting of vertices in $node(3)$, $node(5)$ is the child of $node(3)$. Similarly, according to the relationship of containment, $node(3)$, $node(4)$ are children of $node(2)$, and $node(2)$ is a child of $node(1)$. The distances between users and o_1 are also stored and sorted in each node.

When u_q, k_{min} and c are given in a specific query, we can build a node traversal list L_T for a cd-tree T such that the subtree rooted at each node represents a k-$\widehat{core}$ with at least c vertices and contain u_q for different k. We first search from the node containing u_q, denoted as n_{u_q}. Then we recursively add its ancestor nodes until we find a node with core value no larger than $c-1$ and its subtree contains more than c vertices. Such a node is denoted as the anchor node n_a. We insert this node into the traversal list and recursively put the parent node into the list until the core number of the parent node is smaller than k_{min}. The core number of n_a is the largest possible k value of the (k, c)-cores that we can find.

For example, in Figure 2(b), suppose $u_q = v_1$, $k_{min} = 2$ and $c = 5$. Then the traversal list is $node(5), node(3), node(2)$ whose subtrees represent 4-$\widehat{core}$, 3-$\widehat{core}$, 2-$\widehat{core}$ containing u_q, respectively, and $node(5)$ is the anchor node.

B. Greedy Search of (k, c)-Core

Based on the cd-tree and node traversal list, we illustrate how to find the (k, c)-core with the minimum travel cost greedily, as shown in Algorithm 3. Function GreedyKCCore is adapted from function ExactKCCore in Algorithm 1 with three major differences. First, during this process, users in set V_T are always sorted according to their distances to the meeting point, and we greedily add the nearest user to set V_S. Second, in Algorithm 3, we return the result immediately if we find one group. Finally, in addition to the pruning strategy based on Property 3.4, we add another pruning strategy based on k-core in lines 6 and 11. In line 6, when some users are discarded from V'_T (lines 4 and 5), the size of the k-$\widehat{core}$ containing V'_S may be smaller than c, so expanding set V'_S will not yield the result. Similarly, in line 11, as one vertex is discarded in line 4, we can check the size of k-$\widehat{core}$ and terminate this branch if the size is smaller than c. This technique makes it faster to find the (k, c)-core in the greedy framework. By calling GreedyKCCore for each meeting point, we can get the skyline groups. Since we always add the closest user at each step, the travel cost for the group we finally get is nearly optimal.

Algorithm 4: RSGQ$_\mathsf{D}$-cdtree

```
Input  : G = (G_r, G_s), O, u_q, c, r
Output: Densest road-social group C
1  O' ← find objects by RangeQuery(u_q, r);
2  k_max ← 0;
3  for each o ∈ O' do
4      T_o ← get the cd-tree for object o;
5      T ← T ∪ {T_o};
6  sort T by the core number of the anchor node in each tree;
7  for each T ∈ T do
8      for each node tn ∈ L_T do
9          k ← core number of tn;
10         if k < k_max then
11             if tn is the anchor node return C;
12             else break;
13         V' ← get users in the subtree of tn;
14         V_S = {u_q}; V_T = V' \ {u_q};
15         C = GreedyKCCore(c, k, V_S, V_T)
16         if C ≠ ∅ then
17             k_max = k; maintain the densest groups;
```

V. cd-tree BASED SOLUTIONS

In this section, we show how to answer different RSGQ queries efficiently based on cd-tree.

A. RSGQ$_\mathsf{D}$-cdtree *Algorithm*

Algorithm 4 gives the process of solving the RSGQ$_\mathsf{D}$ problem. Since we focus on the maximal core number, we only need to visit a small part of the cd-tree. In lines 1-4, we get all candidate objects by RangeQuery and then obtain the cd-tree for each object. Next, we sort all the cd-trees based on the core value of each anchor node in line 6 so that we can first access the subtree with a higher core number and get the group with the maximum core number as early as possible. For each tree T, we will check each node in the traversal list to verify the existence of a (k, c)-core in lines 9-17. If k is less than the largest core value we have found, denoted as k_{max}, we will further check whether it is an anchor node. We will terminate the search if it is an anchor node as the core numbers of the anchor nodes in the rest of trees are definitely smaller than current k (line 11), or we will move to the next tree otherwise (line 12). If k is no smaller than k_{max}, we will get the users in this subtree (line 13) and check whether there is a (k, c)-core (line 15). If there is a (k, c)-core, we will update the k_{max} and maintain densest groups (line 17).

Example 5.1. Consider a RSGQ$_\mathsf{D}$ query with $u_q = v_8$, $c = 5$ and $r = 7.0$. First, we use RangeQuery to get candidate

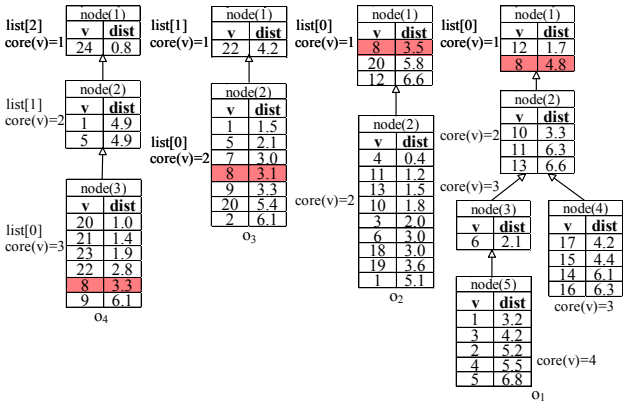


Fig. 3. An example of Algorithm 4

meeting points o_1, o_2, o_3 and o_4 and build the cd-tree for each meeting point as shown in Figure 3. Then objects are sorted as o_4, o_3, o_2, o_1 according to the core value of the anchor node. We first visit $node(3)$ in the tree for o_4, but there does not exist a $(3,5)$-core. Then we visit $node(2)$, and successfully returns group $C_1(\{v_8, v_{20}, v_{21}, v_{22}, v_{23}\}, o_4, 10.4)$ by GreedyKCCore. Thus, k_{max} = 2. Then we visit $node(2)$ of o_3 and get a group $C_2(\{v_1, v_5, v_7, v_8, v_9\}, o_3, 13.0)$. This group is discarded because C_1 has a smaller travel cost. Next, we visit $node(1)$ of o_2, but the core number of $node(1)$ is smaller than k_{max}. Then we terminate the algorithm and return C_1 as the result.

B. RSGQ$_{\mathsf{LS}}$-cdtree *Algorithm*

To get the skyline groups for a given meeting point o, we only need to visit every node in the traverse list and find (k,c)-cores greedily in their subtrees. Algorithm 5 shows the steps of finding the skyline groups by traversing along the cd-tree for object o. From the anchor node, we check each node in the traversal list, find the (k,c)-core level by level, and return the group with the minimum travel cost for different k values by invoking the GreedyKCCore function.

C. RSGQ$_{\mathsf{S}}$-cdtree *Algorithm*

Again, cd-tree enables us to solve the RSGQ$_{\mathsf{S}}$ problem more efficiently, as shown in Algorithm 6. In the beginning, we get all candidate objects by RangeQuery and then obtain the cd-tree for each object. Next, we sort all cd-trees based on the core value of each anchor node. The above process is similar to lines 1-5 in Algorithm 4. Then, for each tree T, we will check each node in the traversal list to see if there is a (k,c)-core and maintain the skyline road-social groups $\mathcal{C}$ in a way similar to lines 2-6 in Algorithm 5. The main difference is that in this algorithm, we need to check multiple trees for a set of meeting points, while in Algorithm 5, we only need to check one tree of a specific meeting point. To further reduce the search space, we will record the smallest travel cost we have obtained for each k, and prune unqualified nodes in the traversal list by the following theorem.

Theorem 5.1. *Given a user u_q, for a node with core number k in the* cd-tree *for meeting point o, if the sum of $dist(o, u_q)$ and other top $c-1$ shortest path distances to o is larger than the lower bound of the travel cost for k, we can skip such node.*

Algorithm 5: RSGQ$_{\mathsf{LS}}$-cdtree

Input : $\mathcal{G} = (G_r, G_s)$, o, u_q, c, r, k_{min}
Output: Skyline road-social groups $\mathcal{C}$

1 $T \leftarrow$ get the cd-tree for object o;
2 **for** *each node* $tn \in L_T$ **do**
3 $k \leftarrow$ core number of node tn;
4 $V_T \leftarrow$ get users in the subtree of tn;
5 $C \leftarrow$ GreedyKCCore$(c, k, \{u_q\}, V_T \setminus \{u_q\})$;
6 maintain the skyline set $\mathcal{C}$ based on C;
7 **return** $\mathcal{C}$;

Algorithm 6: RSGQ$_{\mathsf{S}}$-cdtree

Input : $\mathcal{G} = (G_r, G_s)$, O, u_q, c, r, k_{min}
Output: Skyline road-social groups $\mathcal{C}$

1 lines 1-5 in Algorithm 4;
2 **for** *each* $T \in \mathcal{T}$ **do**
3 lines 2-6 in Algorithm 5;
4 **return** $\mathcal{C}$;

Extension on clique and truss models. The methods proposed in this paper can be easily extended to k-clique and (k,c)-truss models. A k-clique is a clique of size k [18]. The degree of each vertex in this clique is $k-1$, so a k-clique is actually a $(k-1,k)$-core. Thus we can perform $(k-1,k)$-core query by our exact or approximate methods to find k-clique directly. Next, we discuss how to deal with the truss based model in our framework. A k-truss is a connected subgraph where each edge is contained in at least $(k-2)$ triangles [19]. A k-truss is also a $(k-1)$-core, i.e., (k,c)-truss is also a $(k-1,c)$-core. Thus, to get the exact answer for (k,c)-truss based RSGQ problems, we can directly apply all the pruning techniques in the exact method to find the $(k-1,c)$-core and then further check whether it is a (k,c)-truss. Among all the verified (k,c)-trusses, we can return the densest groups or skyline cohesive groups. Moreover, we can also further prune users that cannot form a (k,c)-truss by the bounded diameter property of (k,c)-truss, i.e., the diameter of a connected k-truss with c vertices is bounded by $\lfloor \frac{2c-2}{k} \rfloor$ [19]. For the approximate search for (k,c)-truss based RSGQ problems, we can directly conduct the $(k-1,c)$-core search on cd-tree and then verify whether the $(k-1,c)$-core returned by GreedyKCCore is a (k,c)-truss. Therefore, we can also solve (k,c)-truss based problems under our (k,c)-core framework. As for even more effective solutions, we may exploit the properties of the (k,c)-truss model and build a new index to keep the truss value and distances in a similar way to support efficient truss search.

VI. PERFORMANCE STUDIES

In this section, we report the experimental results to show the efficiency and effectiveness of our proposed algorithm.

A. Experimental Setup

Algorithms. To our best knowledge, there is no previous work that can exactly solve all the three cases of our RSGQ problem. The work most relates to us is CGNN [8], which provides a heuristic solution for the $\mathsf{RSGQ_D}$ problem. We obtained the code from the authors and will report the comparison results in this section. To thoroughly evaluate the proposed techniques, we implemented the following 12 versions of approaches proposed in this paper: 1) three brute-force algorithms at the beginning of Section III, denoted as D-bf, LS-bf, and S-bf, respectively; 2) three efficient exact algorithms in Section III, denoted as D-exact, LS-exact, and S-exact, respectively; 3) three cd-tree based greedy search algorithms in Section V equipped with all the optimization techniques, denoted as D-cd, LS-cd, and S-cd, respectively; 4) three simplified versions of algorithms in Section V where the vertices in V_T are unsorted and randomly chosen in the GreedyKCCore function in Algorithm 3 (line 3), denoted as D-cdr, LS-cdr, and S-cdr, respectively. The RangeQuery in all the algorithms is answered based on the publicly available implementation of G-tree [12]. We generate 100 queries with the same parameter setting and report the average performance. We terminate a query if it takes more than one hour. All the algorithms were implemented in C++, and the experiments were conducted on a Linux server with Intel Xeon E5-2640 (2.6GHZ) processor and 128GB main memory.

Datasets. We use three location-based social-networks (`Brightkite`[2], `Gowalla`[2] and `Foursquare`[3]), and three road networks (`California`[4], `New York`[5], and `Northeast USA`[5]), which are publicly available and widely used in previous works. The details of the datasets are shown in Table I. Since locations in social networks are distributed globally around the world, we normalize the longitude and latitude of the locations of both networks to a two-dimensional [0,1] space and then map users to the nearest intersection or segment in the road network based on their coordinates. We use `BR+CA` to represent the road-social network composed by `Brightkite` and `California`, `CO+NY` to represent `Gowalla` and `New York`, and `FO+NE` to represent `Foursquare` and `Northeast USA`. Moreover, to show the efficiency and effectiveness of our algorithm for real-world cases, we also extracted a sub-network from `Foursquare` consisting of 87,948 users located in `New York`, and denote the largest connected component in this subnetwork as $\mathtt{FO}_{\mathtt{NY}}$ and the combination of social and road networks as $\mathtt{FO}_{\mathtt{NY}}+\mathtt{NY}$. We randomly select $|V(G_s)|/10$ points of interest as candidate meeting points (POI) on the vertices and edges in the road networks. For problem $\mathsf{RSGQ_{LS}}$, we use the nearest POI of the querying user as the meeting point.

Parameters. We consider the effect of 3 parameters, the minimum core number k_{min}, the group size c and the distance threshold r. Here, k_{min} is only evaluated in for problems $\mathsf{RSGQ_{LS}}$ and $\mathsf{RSGQ_S}$, as $\mathsf{RSGQ_D}$ does not require a minimum core number. The ranges of these parameters and their default values are shown in Table II. Similar to the settings in [7], k_{min} is mainly determined by the average degree of the social network, and the default value of c is set to $2k_{min}$. The default value of r is set to about 10 times of the average edge weight for `NY` and `NE`, and 15 times for `CA` due to the small average degree of the corresponding social network `BR`.

[2]http://snap.stanford.edu/data/index.html

[3]https:/archive.org/details/201309_foursquare_dataset_umn

[4]https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

[5]http://users.diag.uniroma1.it/challenge9/download.shtml

TABLE I
DATASETS IN OUR EXPERIMENTS

Social networks	Vertices	Edges	Avg. degree
BrightKite	58,228	214,078	7.35
Gowalla	196,591	950,327	9.67
Foursquare	2,153,471	13,549,245	12.58
Road networks	Vertices	Edges	Avg. edge weight
California	21,048	21,693	0.016186
New York	264,346	733,846	1293.3
Northeast USA	1,524,453	3,897,636	1714.9

TABLE II
PARAMETER SETTINGS

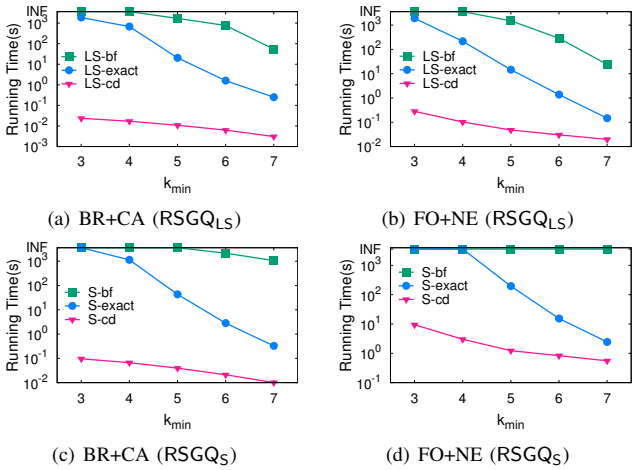
Parameters	Ranges	Defaults
k_{min}	3, 4, 5, 6, 7	5
c	8, 9, 10, 11, 12	10
r (`CA`)	0.15, 0.20, 0.25, 0.30, 0.35	0.25
r (`NY`)	9000, 12000, 15000, 18000, 21000	15000
r (`NE`)	12000, 16000, 20000, 24000, 28000	20000

B. Efficiency Evaluation

Varying k_{min}. Figure 4 reports the running time of all the algorithms for problems $\mathsf{RSGQ_{LS}}$ and $\mathsf{RSGQ_S}$ when varying k_{min}. We can see that as k_{min} increases, all of our algorithms become faster. It is because the number of (k, c)-cores will be reduced substantially since many subgraphs cannot satisfy the large social constraint. When k_{min} is small, under the constraint of default c, the pruning techniques become less powerful, especially when $c \geq 2k_{min} + 2$. The reason is that when $c \geq 2k_{min} + 2$, Property 3.4 is not applicable, and the diameter bound is not as tight as that for $c < 2k_{min} + 2$ (Property 3.3). As expected, LS-cd and S-cd perform better than the exact algorithms and brute-force algorithms by 1-5 order of magnitude, especially when k_{min} is small. The reason is that the k_{min}-$\widehat{core}$ extracted in the social network is larger for a smaller k_{min}, which leads to large enumeration space in the brute-force and exact algorithms.

Varying c. Figure 5 shows the running time for all the three problems when we vary c. As c increases, all the algorithms need more time because more combinations of the vertices need to be considered. Especially when $c \geq 2k_{min} + 2$, the exact algorithms become much slower, and the brute-force algorithms for one single query cannot even finish in one hour. As expected, our cd-tree based algorithms are still very fast as we successfully avoid the explosive enumeration of c users.

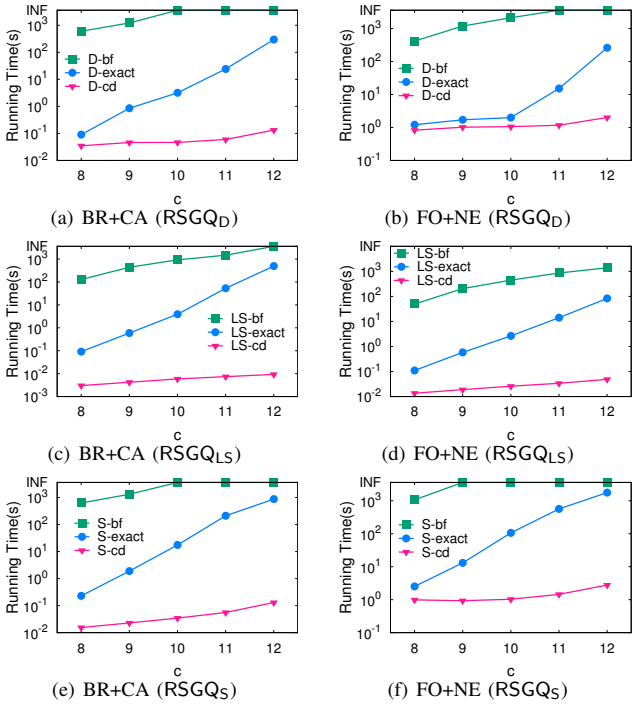
Varying r. Figure 6 reports the running time for all the three problems with respect to r. Overall, all algorithms take more time as the distance threshold r increases, because more candidate meeting points and users are included for a larger



(a) BR+CA ($\mathsf{RSGQ}_{\mathsf{LS}}$) (b) FO+NE ($\mathsf{RSGQ}_{\mathsf{LS}}$)

(c) BR+CA ($\mathsf{RSGQ}_{\mathsf{S}}$) (d) FO+NE ($\mathsf{RSGQ}_{\mathsf{S}}$)

Fig. 4. Varying k_{min}



(a) BR+CA ($\mathsf{RSGQ}_{\mathsf{D}}$) (b) FO+NE ($\mathsf{RSGQ}_{\mathsf{D}}$)

(c) BR+CA ($\mathsf{RSGQ}_{\mathsf{LS}}$) (d) FO+NE ($\mathsf{RSGQ}_{\mathsf{LS}}$)

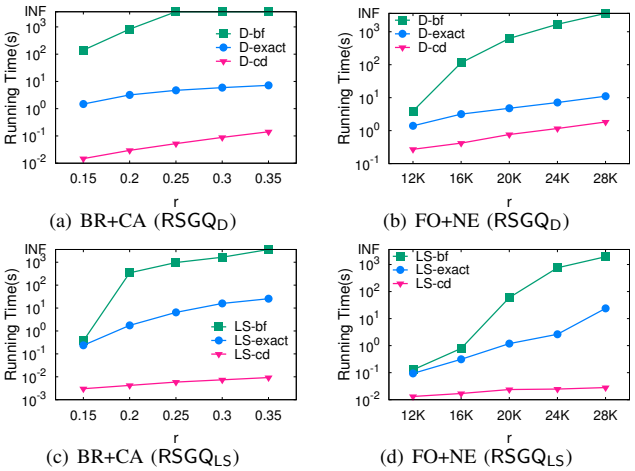
(e) BR+CA ($\mathsf{RSGQ}_{\mathsf{S}}$) (f) FO+NE ($\mathsf{RSGQ}_{\mathsf{S}}$)

Fig. 5. Varying c

range of r. Again, our cd-tree based greedy search algorithms outperform exact algorithms and brute-force algorithms significantly on all the tasks.

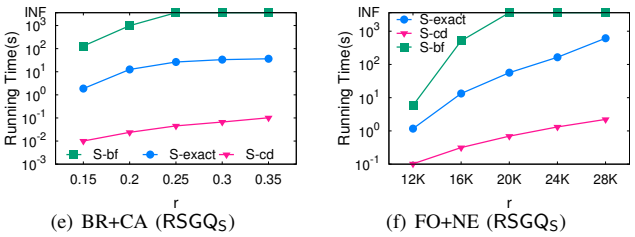
C. Effectiveness Evaluation

Effectiveness of greedy strategy. We evaluate the effectiveness of our greedy strategy that selects vertices based on the distance by comparing methods D-cd, LS-cd and S-cd with D-cdr, LS-cdr, and S-cdr. Since our cd-tree based algorithms can find the (k, c)-core groups for each specific k, the error of the core number is 0. Thus we will only report the error of travel cost. For each group returned by the exact algorithm, we find the corresponding group returned by the cd-tree based algorithm under the same input. Then the cost error ξ_{cost} is



(a) BR+CA ($\mathsf{RSGQ}_{\mathsf{D}}$) (b) FO+NE ($\mathsf{RSGQ}_{\mathsf{D}}$)

(c) BR+CA ($\mathsf{RSGQ}_{\mathsf{LS}}$) (d) FO+NE ($\mathsf{RSGQ}_{\mathsf{LS}}$)



(e) BR+CA ($\mathsf{RSGQ}_{\mathsf{S}}$) (f) FO+NE ($\mathsf{RSGQ}_{\mathsf{S}}$)

Fig. 6. Varying r

TABLE III
APPROXIMATION OF THE TRAVEL COST ξ_{cost}

	Algorithms					
DataSets	D-cd	D-cdr	LS-cd	LS-cdr	S-cd	S-cdr
BR+CA	**1.75%**	9.35%	**4.50%**	22.64%	**3.86%**	21.60%
GO+NY	**0.39%**	6.81%	**1.15%**	16.51%	**0.97%**	16.80%
FO+NE	**0.54%**	6.50%	**2.60%**	16.45%	**1.45%**	12.58%

defined as the average error of all groups returned by the exact algorithms. Table III shows the travel cost error of our greedy algorithms, where D-cd, LS-cd and S-cd only yield small errors since we always choose the closest user greedily, while D-cdr, LS-cdr, and S-cdr generate larger error as the users are randomly selected at each step.

Comparison with the state-of-the-art algorithms. We compare our D-cd algorithm with the state-of-the-art algorithm CGNN [8] proposed for the $\mathsf{RSGQ}_{\mathsf{D}}$ problem on GO+NY. As discussed in Section III-A, CGNN may fail to return any group because users of a group are determined before considering locations of meeting points. Thus, we only evaluate the queries with non-empty results in CGNN. As shown in Table IV, the average core number found by CGNN is generally around 5 with more travel cost and running time of CGNN, while our algorithm can return the exact core number around 6. This validates our previous analysis that CGNN may not find the group with the maximum core number and our method can find group with the maximum core number with relatively higher spatial cohesiveness.

D. Case study on road-social network in New York $FO_{NY}+NY$

Now, we evaluate our algorithms by a real-world case study on the road-social network in New York $\mathtt{FO}_{\mathtt{NY}} + \mathtt{NY}$.

First of all, we show the efficiency of our algorithms in terms of running time, as shown in Table V. Since the size

TABLE IV
COMPARISON WITH THE STATE-OF-THE-ART ALGORITHMS

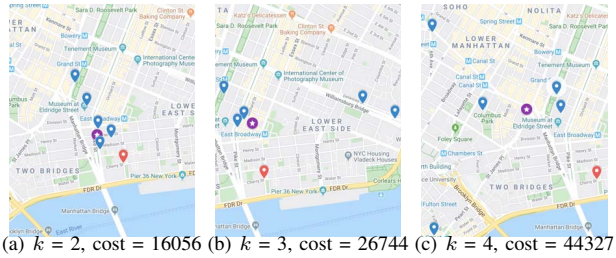
Metrics	Algorithms	$c = 8$	$c = 9$	$c = 10$	$c = 11$	$c = 12$
avg. k	CGNN	4.556	5.010	4.960	4.755	4.835
	D-cd	**5.929**	**6.260**	**6.370**	**6.439**	**6.546**
ξ_{cost}	CGNN	35.7%	38.6%	34.40%	34.02%	25.9%
	D-cd	**0.42%**	**1.12%**	**0.26%**	**0.41%**	**0.97%**
Time (s)	CGNN	0.2026	0.2101	0.2154	0.2271	0.2283
	D-cd	**0.0711**	**0.0747**	**0.0933**	**0.1092**	**0.1188**

TABLE V
RUNNING TIME ON $\text{FO}_{\text{NY}} + \text{NY}$ (S)

Algorithms	$c = 5$	$c = 6$	$c = 7$	$c = 8$	$c = 9$
D-bf	3.8823	15.686	98.529	171.94	267.53
D-exact	0.2634	0.4937	3.2639	6.5346	17.810
D-cd	**0.0515**	**0.0525**	**0.0526**	**0.0545**	**0.0610**
LS-bf	0.0021	0.0136	0.0308	14.601	38.639
LS-exact	0.0018	0.0079	0.0252	0.1139	0.4298
LS-cd	**0.0012**	**0.0016**	**0.0017**	**0.0018**	**0.0020**
S-bf	3.4118	32.529	203.10	651.12	INF
S-exact	0.0275	0.2557	3.0542	51.557	186.22
S-cd	**0.0294**	**0.0342**	**0.0345**	**0.0347**	**0.0348**

and density of FO_{NY} are small, we set k_{min} = 3. We set r = 10000 so that the brute-force algorithms can finish in one hour in most cases. We vary c to test the performance of the algorithms. Similar to the trend in Figure 5, the running time of all the algorithms increases as c becomes larger, and our cd-tree based methods are faster than the brute-force methods and exact methods by 2-5 orders of magnitude. Note that although this dataset is not very large, brute-force and exact methods for large c still take hundreds of seconds sometimes, while the cd-tree based methods always take less than 0.1 seconds.

We also evaluate the effectiveness of our skyline model by performing a RSGQ_S query for user u_q = 530, and set k_{min} = 2, c = 6, and r = 10000. Three skyline groups returned are presented in Figure 7. In each subfigure, u_q is marked by a red pin, other users are marked blue pins, and the meeting point is marked by a purple star. We can see that from Figure 7(a) to 7(c), as k increases, the travel cost increases. This validates the skyline property of these two dimensions and shows that our algorithms for RSGQ_S can return a set of skyline road-social groups for users to choose, which can successfully balance the social cohesiveness and spatial cohesiveness.



(a) k = 2, cost = 16056 (b) k = 3, cost = 26744 (c) k = 4, cost = 44327

Fig. 7. Case study on $\text{FO}_{\text{NY}} + \text{NY}$

VII. RELATED WORK

Group queries on social networks. Group query/community search on social networks aims to retrieve a dense subgraph containing a querying user. Dense subgraph models include clique and quasi-clique [18] [21] [20], k-core [22] [23] [24], k-truss [25] [26] [27], and so on. Some recent works mainly focus on searching groups in attributed social networks. For example, Li et al. [28] studied the skyline group search on graphs with numeric attributes. [17] and [29] considered both keyword and structure cohesiveness when searching groups. [30] and [31] studied the group search on weighted networks to obtain groups with both strong influence and cohesive structure. However, all of these works did not consider road information and cannot solve the problem studied in this paper. Note that although the coordinates can be considered the numeric attributes of multi-valued networks, our target is to optimize the travel cost which is previously unknown. Thus, our work is substantially different from the skyline query on multi-valued networks [28].

Group queries on spatial databases. Group nearest neighbor query, denoted as GNN query, finds the meeting point(s) that minimize the sum of distances from a given set of locations [32]. [33] extends the work in [32] by generalizing the sum of distances to an aggregate function (ANN). [34] further extends ANN from euclidean space to road networks. In recent years, the GNN query has also been widely on other kinds of graphs, such as uncertain graphs [35], [36], private graphs [37], [38] and graphs with moving objects [39] or query points [40]. However, all the above works did not take the social cohesiveness into consideration.

Group queries on geo-social networks. Geo-social group queries aim to find the subgraph that is cohesive in the social network and close in space. Yang et al. [1] considered a socio-spatial group query to minimize the total spatial distance to a provided meeting point, and each member is allowed to be unfamiliar with at most a maximum number of group members. [3] provides a set of geo-social ranking functions that consider both social and spatial information and find top-k users for each of these functions. [2] finds a meeting point and the corresponding best group which has the minimum travel cost to the meeting point and satisfies the average minimum familiarity constraint.

Here, we elaborate on the recent works most related to ours, which share the same or similar structure models, i.e., k-core [4]–[8], [10] and k-truss [9]. For the spatial constraint, [4] and [5] require users of a group to be inside a circle, and [10] and [9] require users' pairwise distances to be smaller than a given threshold. All of these four algorithms aim to find groups whose size is maximal/minimal. On the other hand, [6]–[8] aims to minimize the travel cost to some specific meeting points with a certain requirement of the group size. In [6], Zhu et al. designed an offline index to answer such queries, but their index can not be adapted to road network distances, and they only find one group for each query. [7] provides a ranking function combining the social cohesiveness and spatial closeness, but it requires some predefined parameters which are hard to tune. [8] is the first work on cohesive group queries on a road-social network, where their problem can be considered as a special case of ours. Moreover, they only provided a heuristic solution that cannot guarantee the return of the exact results as we have discussed before. To our best knowledge, our work is the first to study the skyline

TABLE VI
SUMMARY OF RELATED WORKS AND OUR WORK

	[4]	[5]	[10]	[9]	[6]	[7]	[8]	**our work**
social constraint	k-core	k-core	k-core	k-truss	k-core	k-core	k-core	**(skyline)** k-core
spatial constraint	radius	radius	pairwise distance	pairwise distance	range & cost	cost	cost	cost
spatial model	euclidean	euclidean	euclidean	euclidean	euclidean	euclidean	road network	road network
community size	minimal	maximal	maximal	maximal	fixed	range	fixed	fixed

road-social group search problem, and exact and approximate solutions are both provided to solve this problem.

VIII. CONCLUSION

In this paper, we study the skyline cohesive group query problem on road-social networks. We propose a novel group search model, skyline road-social group, in which each returned group is not dominated by any other group with respect to the social and spatial cohesiveness. We develop efficient exact algorithms to find groups in the context of different problems. We also design cd-tree to allow core-aware greedy algorithms with minor error ratios. Extensive experiments demonstrate the efficiency and effectiveness of our algorithms.

Acknowledgment The work was supported in part by grants of Natural Science Foundation 61972291, Natural Science Foundation of Hubei Province 2018CFB519, Fundamental Research Funds for the Central Universities 413000078, Research Grant Council of the Hong Kong SAR 14202919 and 14203618. We also would like to thank Fangda Guo et al. [8] for providing the code of CGNN. Yuanyuan Zhu is the corresponding author.

REFERENCES

[1] D. Yang, C. Shen, W. Lee, and M. Chen, "On socio-spatial group query for location-based social networks," in *KDD*, 2012, pp. 949–957.
[2] C. Shen, D. Yang, L. Huang, W. Lee, and M. Chen, "Socio-spatial group queries for impromptu activity planning," *TKDE*, vol. 28, no. 1, pp. 196–210, 2016.
[3] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *PVLDB*, vol. 6, no. 10, pp. 913–924, 2013.
[4] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *PVLDB*, vol. 10, no. 6, pp. 709–720, 2017.
[5] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin, "Efficient computing of radius-bounded k-cores," in *ICDE*, 2018, pp. 233–244.
[6] Q. Zhu, H. Hu, C. Xu, J. Xu, and W.-C. Lee, "Geo-social group queries with minimum acquaintance constraints," *VLDBJ*, vol. 26, no. 5, pp. 709–727, 2017.
[7] B. Ghosh, M. E. Ali, F. M. Choudhury, S. H. Apon, T. Sellis, and J. Li, "The flexible socio spatial group queries," *PVLDB*, vol. 12, no. 2, pp. 99–111, 2018.
[8] F. Guo, Y. Yuan, G. Wang, L. Chen, X. Lian, and Z. Wang, "Cohesive group nearest neighbor queries over road-social networks," in *ICDE*, 2019, pp. 233–244.
[9] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, "Maximum co-located community search in large scale social networks," *PVLDB*, vol. 11, no. 10, pp. 1233–1246, 2018.
[10] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "When engagement meets similarity: Efficient (k,r)-$core$ computation on social networks," *PVLDB*, vol. 10, no. 10, pp. 998–1009, 2017.
[11] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *arXiv*, vol. cs.DS/0310049, 2003.
[12] R. Zhong, G. Li, K. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *TKDE*, vol. 27, no. 8, pp. 2175–2189, 2015.
[13] Z. Li, L. Chen, and Y. Wang, "G*-tree: An efficient spatial index on road networks," in *ICDE*, 2019, pp. 268–279.
[14] S. B. Seidman and B. L. Foster, "A graphtheoretic generalization of the clique concept," *The Journal of Mathematical Sociology*, vol. 6, no. 1, pp. 139–154, 1978.
[15] A. Conte, T. D. Matteis, D. D. Sensi, R. Grossi, A. Marino, and L. Versari, "D2K: scalable community detection in massive networks via small-diameter k-plexes," in *SIGKDD*, 2018, pp. 1272–1281.
[16] U. Brandes and T. Erlebach, Eds., *Network Analysis: Methodological Foundations*, ser. Lecture Notes in Computer Science, vol. 3418. Springer, 2005.
[17] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu, "Effective and efficient attributed community search," *VLDB J.*, vol. 26, no. 6, pp. 803–828, 2017.
[18] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013, pp. 277–288.
[19] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *National Security Agency Technical Report*, vol. 16, 2008.
[20] P. Lee and L. V. S. Lakshmanan, "Query-driven maximum quasi-clique search," in *ICDM*, 2016, pp. 522–530.
[21] M. Wang, C. Wang, J. X. Yu, and J. Zhang, "Community detection in social networks: An in-depth benchmarking study with a procedure-oriented framework," *PVLDB*, vol. 8, no. 10, pp. 998–1009, 2015.
[22] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *SIGMOD*, 2014, pp. 991–1002.
[23] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *SIGKDD*, 2010, pp. 939–948.
[24] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *DMKD*, vol. 29, no. 5, pp. 1406–1433, 2015.
[25] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *SIGMOD*, 2014, pp. 1311–1322.
[26] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *PVLDB*, vol. 9, no. 4, pp. 276–287, 2015.
[27] E. Akbas and P. Zhao, "Truss-based community search: a truss-equivalence based indexing approach," *PVLDB*, vol. 10, no. 11, pp. 1298–1309, 2017.
[28] R. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, "Skyline community search in multi-valued networks," in *SIGMOD*, 2018, pp. 457–472.
[29] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *PVLDB*, vol. 10, no. 9, pp. 949–960, 2017.
[30] Z. Zheng, F. Ye, R. Li, G. Ling, and T. Jin, "Finding weighted k-truss communities in large networks," *Information Science*, vol. 417, pp. 344–360, 2017.
[31] R. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *PVLDB*, vol. 8, no. 5, pp. 509–520, 2015.
[32] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *ICDE*, 2004, pp. 301–312.
[33] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *TODS*, vol. 30, no. 2, pp. 529–576, 2005.
[34] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *TKDE*, vol. 17, no. 6, pp. 820–833, 2005.
[35] X. Lian and L. Chen, "Probabilistic group nearest neighbor queries in uncertain databases," *TKDE*, vol. 20, no. 6, pp. 809–824, 2008.
[36] Z. Liu, C. Wang, and J. Wang, "Aggregate nearest neighbor queries in uncertain graphs," *WWW*, vol. 17, no. 1, pp. 161–188, 2014.
[37] Y. Wu, K. Wang, Z. Zhang, W. Lin, H. Chen, and C. Li, "Privacy preserving group nearest neighbor search," in *EDBT*, 2018, pp. 277–288.
[38] T. Hashem, L. Kulik, K. Ramamohanarao, R. Zhang, and S. C. Soma, "Protecting privacy for distance and rank based group nearest neighbor queries," *WWW*, vol. 22, no. 1, pp. 375–416, 2019.
[39] H. G. Elmongui, M. F. Mokbel, and W. G. Aref, "Continuous aggregate nearest neighbor queries," *GeoInformatica*, vol. 17, no. 1, pp. 63–95, 2013.
[40] J. Li, J. R. Thomsen, M. L. Yiu, and N. Mamoulis, "Efficient notification of meeting points for moving groups via independent safe regions," *TKDE*, vol. 27, no. 7, pp. 1767–1781, 2015.