# Continuous Geo-Social Group Monitoring over Moving Users

Huaijie Zhu, Wei Liu*, Jian Yin
*Sun Yat-sen University, China*
*Laboratory of Big Data Analysis and Processing, Guangzhou*
zhuhuaijie, liuw259, issjyin@mail.sysu.edu.cn

Mengxiang Wang
*Northeastern University, China*
wmx0425@gmail.com

Jianliang Xu
*Hong Kong Baptist University*
xujl@comp.hkbu.edu.hk

Xin Huang
*Hong Kong Baptist University*
xinhuang@comp.hkbu.edu.hk

Wang-Chien Lee
*The Pennsylvania State University, USA*
wlee@cse.psu.edu

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00028

***Abstract*—Recently a lot of research works have focused on *geo-social group* queries for group-based activity planning and scheduling in location-based social networks (LBSNs), which return a social cohesive user group with a spatial constraint. However, existing studies on geo-social group queries assume the users are stationary whereas in real LBSN applications all users may continuously move over time. Thus, in this paper we investigate the problem of *continuous geo-social groups monitoring* (*CGSGM*) over moving users. A challenge in answering CGSGM queries over moving users is how to efficiently update geo-social groups when users are continuously moving. To address the CGSGM problem, we first propose a baseline algorithm, namely *Baseline-BB*, which recomputes the new geo-social groups from scratch at each time instance by utilizing a branch and bound (BB) strategy. To improve the inefficiency of BB, we propose a new strategy, called common neighbor or neighbor expanding (CNNE), which expands the common neighbors of edges or the neighbors of users in intermediate groups to quickly produce the valid group combinations. Based on CNNE, we propose another baseline algorithm, namely *Baseline-CNNE*. As these baseline algorithms do not maintain any intermediate results to facilitate further query processing, we develop an incremental algorithm, called *incremental monitoring algorithm (IMA)*, which maintains the support, common neighbors and the neighbors of current users when exploring possible user groups for further updates and query processing. Finally, we conduct extensive experiments using three real datasets to validate our ideas and evaluate the proposed algorithms.**

***Index Terms*—geo-social group queries, continuous queries, moving users**

## I. Introduction

With the ever-growing popularity of GPS-enabled devices and online social networks, location-based social networks (LBSNs), e.g., Foursquare, Yelp, Wechat, and Weibo, have emerged in our social life. In all these LBSNs, mobile users are allowed to share their check-in locations (e.g., homes, supermarkets, offices, restaurants, and shopping malls) with friends or social users. LBSNs have bridged the gap between the physical world and the virtual world of social networks, providing social users new applications, such as group-based activity planning and target marketing [1]–[5]. For example, a hotpot restaurant would like to push group coupons for nearby users, who are socially connected with each other, so as to attract the users to dine at the restaurant. Accordingly, a geo-social group query may be issued to find a socially cohesive group of users with a spatial constraint for the application [1].

While much research attention has recently been drawn to geo-social group queries (e.g., [1], [2], [4]–[6]), existing works assume that users' locations are fixed. Nevertheless, we argue that users could be moving in real-life scenarios. In other words, the user groups satisfying the query conditions may constantly change over time. In the above restaurant example, group coupons should be advertised to periodically detected user groups near the restaurant.

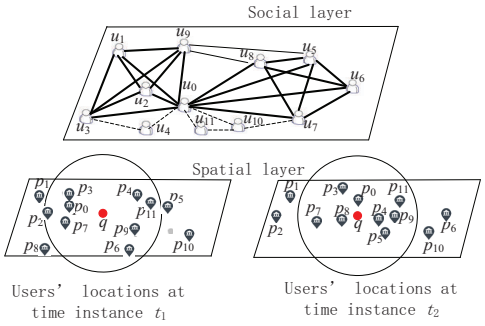



Figure 1. An example of CGSGM query

To make geo-social queries more realistic and useful for location-based applications, in this paper, we introduce a new geo-social group query, namely, *continuous geo-social groups monitoring query over moving users* (CGSGM).[1] Consider an LBSN represented by a graph $G$. Given a CGSGM query, specified in the form $\langle d, q_l, k, p, [t_1, t_2]\rangle$, the system continuously monitors all user groups of size $p$ such that the distance of any result user to the query location $q_l$ is not greater than $d$ and that the users within a group form a $k$-truss (i.e., a subgraph in which each edge $e(u, v)$ has at least $k$-2 common neighbors). An example of a CGSGM query is illustrated in Figure 1,

[1]We assume that the social network does not change during the query period, which may last for a few hours.

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00028



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

where the social network and user locations are respectively presented in a social layer and a spatial layer. The top part is the social graph where two users are connected if two users are acquainted with each other, while the bottom part shows users' locations at time instances $t_1$ and $t_2$, respectively, where each user $u_i$ is associated with its location $p_i$. Suppose there is a restaurant located at $q$, which aims to continuously identifying all social cohesive groups of size 5 whose distance to the restaurant is not greater than 2km, during the opening hours $[t_1$ = 5pm, $t_2$ = 9pm]. This task can be carried out by issuing a CGSGM query $\langle d$ = 2km, $q$, $k$ =3, $p$=5, $[t_1, t_2]\rangle$. The results of this CGSGM query are the groups $\{u_0, u_2, u_3, u_4, u_9\}$, $\{u_0, u_2, u_3, u_6, u_7\}$, $\{u_0, u_2, u_6, u_7, u_9\}$, $\{u_0, u_3, u_4, u_6, u_7\}$, and $\{u_0, u_3, u_9, u_6, u_7\}$ for time instance $t_1$, while the group results for time instance $t_2$ are $\{u_0,\ u_3, u_4, u_7, u_5\}$, $\{u_0, u_3, u_4, u_9, u_5\}$, $\{u_0, u_3, u_9, u_7, u_5\}$, $\{u_0, u_3, u_4, u_5, u_8\}$, $\{u_0, u_3, u_4, u_7, u_8\}$, $\{u_0, u_3, u_9, u_5, u_8\}$, $\{u_0, u_3, u_9, u_7, u_8\}$, and $\{u_0, u_5, u_7, u_9, u_8\}$.

To support CGSGM, we propose a query processing framework (as shown in Figure 2). We assume that there is a central server that receives the positions of mobile users from the position monitoring subsystem. The CGSGM queries issued by query clients (e.g., various restaurants) are registered at this central server, where query processor continuously returns query results to the query clients, upon reception of user position updates. We propose efficient algorithms to process CGSGM queries in real time. The server does not assume any knowledge about the users' moving velocities, directions, or trajectories.

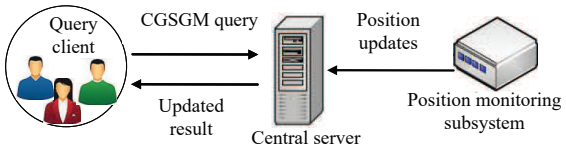



Figure 2. The CGSGM framework

A major challenge faced in processing CGSGM queries is the need to enumerate every possible group combination to generate the result groups at each time instance. It is easy to prove that the CGSGM query problem is NP-hard.

To tackle the CGSGM problem, we first propose a baseline, which recomputes new geo-social groups from scratch at each time instance. To find all $k$-truss groups of size $p$, a basic idea is to utilize the branch and bound (BB) method to numerate all the possible $k$-truss groups of size $p$ with trussness filtering and $k$-truss decomposition. As the BB method fetches users one by one without exploiting some specific orders and social constraints to form the groups, it is hard to produce a feasible solution. Thus, we propose a *common neighbor or neighbor expanding* (CNNE) strategy, by expanding the common neighbors of edges or the neighbors of users in an intermediate group at each time, to quickly produce valid group combinations. In addition, we develop an edge support based deciding (ESD) scheme to decide which neighbor or common neighbor to choose first in the CNNE strategy in order to expand the current branch. Accordingly, based on the two expanding strategies (i.e., BB and CNNE), we develop two baseline algorithms, namely *Baseline-BB* and *Baseline-CNNE*, which recompute new geo-social groups from scratch at each time instance.

Through preliminary testing of those baseline algorithms, we find the support, the common neighbors, and the neighbors of the candidate users need to be calculated over and over when new updates occur. Therefore, we further propose an efficient incremental monitoring algorithm (IMA), by maintaining these useful information and intermediate results for further query processing. IMA includes two parts: initial result computation and multiple-users update. When exploring possible $k$-truss groups at each time instance of query processing, we maintain the support, the common neighbors, and the neighbors of current users for future user updates and query processing. Finally, we conduct a comprehensive performance evaluation using three real datasets to validate our ideas and compare the proposed algorithms.

The contributions made in this paper are four-fold:

1. We formalize a new and realistic variant of geo-social query, *continuous geo-social groups monitoring* (*CGSGM*) query, over moving users. To the best of our knowledge, this is the first attempt to tackle the CGSGM problem.

2. We propose two baseline algorithms, namely Baseline-BB and Baseline-CNNE, which recompute new geo-social groups from scratch at each time instance.

3. We further develop an efficient incremental algorithm, which maintains useful intermediate results for further user updates and query processing.

4. We conduct extensive experiments to evaluate the proposed algorithms. The experimental results show that IMA significantly outperforms the two baseline algorithms.

The rest of this paper is organized as follows. Section II reviews the related work. Section III gives some basic definitions and formulates the CGSGM problem. Section IV presents our two baseline algorithms in detail. Section V introduces the incremental monitoring algorithm. Section VI reports the experimental results and our findings. Finally, Section VII concludes the paper with a discussion of future work.

## II. Related Work

A lot of research efforts related to our work, including *Continuous Spatial Queries*, and *Geo-Social Group Queries*, have been made in the field of location based social network.

### A. Continuous spatial queries (CSQs)

Many continuous spatial queries over spatial databases have been studied over the years.

Since 1990s, researchers have been studying the problem of querying moving objects [7]. Queries such as "find Alices nearest petrol stations while she is driving" or "find all the taxi cabs within 1 km from Alice" [8] are proposed for research. As technology advances, mobile devices emerge in our daily life, reserach attention on continuous spatial queries (CSQs) also grows significantly in the spatial database



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

community. A large number of studies are investigated from various aspects of CSQs, including access methods [9]–[11], query algorithms [12]–[14] and new query types [15]–[17], just to name a few. Over the years, a CSQ has also been called as a moving query [17]–[19], an active query [20], a (continuous) spatio-temporal query [9], [21], [22], and a continuous location-based (or location-dependent) query [23]–[26], with some minor differences in the targeted settings. The terms moving query, spatio-temporal query, and active query are introduced in part to distinguish the new functionality from that of traditional spatial queries where the objects are static and time-independent. The terms continuous location-based or location-dependent query are introduced in part to emphasize the application context of location-based service (LBS). Xiong et al. [27] study the processing of continuous $k$ nearest neighbor queries in spatio-temporal databases. In their setting, both the objects and continuous queries may change their locations over time. The answer region of a query point $q$ is defined as the circle centered at $q$ wtih radius $best\_dist$, which is the distance of the $k$th NN to $q$. Mouratidis et al. [28] propose the conceptual partitioning monitoring (CPM) method for continuous nearest neighbor monitoring. They define the influence region of $q$ as the set of cells that intersect the circle centered at $q$ with radius $best\_dist$. Only the updates affecting the influence region of a query are used to invalidate its current result. However, like many other existing works, the two works mentioned above do not consider the social constraint in query processing.

### B. Geo-social group queries

Various location-based social network sites allow users to share their locations through check-ins or mentions in posts. Over the years, the prosperity of location-based social networking paves the way for new applications of group-based activity planning and marketing. As a result, a growing number of researches on geo-social (socio-spatial) group queries, have been explored.

For a given geo-social graph, Yang et al. [1] first propose a geo-social group query (SSGQ) that finds a set of members based on a fixed rally point where the aggregated spatial distance between members and the rally point is minimized while each member is allowed to be unfamiliar with at most a given number of members in that group. However, such a social constraint may lead to a group that have very distant or diverse social relations. Thus, Zhu et al. [2] propose a new class of geo-social group queries with *minimum acquaintance constraint* (GSGQ), which ensures all users in the result group to have at least a certain number of acquaintances. GSGQ takes three parameters: query issuer, spatial constraint, and social constraint, where the query issuer is a member in the given graph. Although SSGQ and GSGQ impose a constraint on group size, they do not consider user movements. Thus, the query results are static. Recently, the geo-social $k$-cover group query for collaborative spatial computing is proposed [5]. In this query, given a set of query points and a social network, it retrieves a minimum user group in which each user is socially related to at least $k$ other users such that the user-associated regions (e.g., familiar regions or service regions) can jointly cover all the query points. In addition, Armenatzoglou et al. [29] propose a general framework for geo-social query processing, which separates the social, geographical, and query processing modules to facilitate flexible data management. Then, Shen et al. [4] propose the multiple rally-point social spatial group query (MRGQ) that chooses a suitable rally point from the multiple points and the corresponding best group, by minimizing spatial distance between group members to the best rally point. Fang et al. [30] propose a spatial-aware community (SAC), which is a connected $c$-core where the members in the resulting group are located within a spatial circle of minimum radius. SAC also maintains the minimum acquaintance constraint. Furthermore, Shen et al. [31] investigate the problem of computing the radius-bounded $k$-cores (RB-$k$-cores) that aims to find cohesive subgraphs satisfying both social and spatial constraints on large geo-social networks. By considering the constraint of users' spatial information in $k$-truss search, Chen et al. [32] study the co-located community search to find the maximum co-located communities. Although this work also uses $k$-truss to measure the community, its goal is to find the maximum community. The flexible socio-spatial group queries [33] are proposed to find the top $k$ groups w.r.t. multiple POIs where each group follows the minimum social connectivity constraint. In addition, Li et al. [34] propose 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. Finally, Chen et al. [35] propose a novel geo-social group model, equipped with elegant keyword constraints.

In summary, the problem settings of the aforementioned existing works are different from CGSGM. In particular, they do not study the dynamic change of the query results caused by the movements of users, which is the main focus of this work. Moreover, the size of geo-social groups returned by CGSGM is fixed, which introduces significant challenges to our work.

## III. PRELIMINARIES

Given a location-based social network (LBSN), we focus on continuously finding geo-social groups of moving users located in monitored spatial regions, where each user may be continuously moving. In this paper, we assume an environment where each of the moving objects is equipped with a location detection device, e.g., a smart phone with GPS. The moving users report their locations periodically. With respect to the spatial constraint, we denote the Euclidean distance between a user $u$ and a query location $q_l$ by $dis(u, q_l)$. For continuous evaluation of spatio-temporal queries on moving users, where user locations may change constantly, two models may be considered: (1) updates are pushed into the query processor as soon as they are available (e.g., as in [14]); (2) the query processor periodically pulls the current locations of the objects for query processing in order to update the query answer [36]. In this paper, we adopt the second model and assume that the query processor pulls the current locations of the objects at a



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

scheduled time instance to generate/update the query answer for the query issuer[2].

In order to find a social cohesive group of users in a social network, similar to other related works on geo-social network, CGSGM uses $k$-truss [37] as the basis of social constraint to restrict the result group. In this section, we first introduce the related definitions and properties of $k$-truss, based on which the CGSGM problem is formulated.

Consider an undirected graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. A triangle in $G$ is a cycle of length 3. Let $u, v, w \in V$ be the three vertices on the triangle, denoted by $\triangle uvw$. We first define the notion of *common neighbor* and *support* of an edge, respectively.

*Definition 1:* (COMMON NEIGHBOR). Given an edge $e(x, y)$, we say $z$ is the common neighbor to $e(x, y)$ iff $e(x, z)$ and $e(y, z) \in E$. In other words, $x, y, z$ form a triangle.

*Definition 2:* (SUPPORT). The support of an edge $e(u, v) \in E$ in $G$, denoted by $sup(e, G)$, is defined as $|\{\triangle uvw : w \in V\}|$. When the context is obvious, we replace $sup(e, G)$ by $sup(e)$. Since $w$ is a common neighbor of $(u, v)$, the support of $e(u, v)$ is the total number of common neighbors to $e(u, v)$.

With the definition of support, $k$-truss [38] is defined as follows. Intuitively, a $k$-truss is subgraph in which each edge $e(u, v)$ has at least $k$-2 common neighbors. Let $G[W]$ denote a subgraph induced by a group set $W \subseteq V$.

*Definition 3:* ($k$-TRUSS GROUP.) Given a group set $W$ and an integer $k \geq 2$, $W$ is a $k$-truss group, if there exist at least one induced subgraph $G[W]$ satisfying the following two conditions:

(1) **$k$-truss**. $G[W]$ is a subgraph of $G$, denoted as $G[W] \subseteq G$, such that $\forall e \in E(G[W])$, $sup(e, G[W]) \geq (k - 2)$;

(2) **Connected**. $G[W]$ is connected.

For example, as shown in Figure 1, group $\{u_0, u_1, u_2, u_9, u_7, u_8, u_5\}$ is 4-truss group. Notice that the support of edges in its induced graph are not less than 2 except the edge $e(u_5, u_9)$ whose support is 1. By removing edge $e(u_5, u_9)$, the subgraph is still is connected and satisfied as a 4-truss. Based on the observation, we devise an algorithm to check whether a group is $k$-truss. The main idea is to remove an edge whose support is less than $k$-2 iteratively until we find the induced subgraph is a $k$-truss or at least one node is disconnected from other nodes. That is, if there exists one node disconnected from other nodes, such a group is not a $k$-truss group.

To facilitate our discussion, we define the *trussness* of a subgraph, an edge, and a vertex as follows.

*Definition 4:* (SUBGRAPH TRUSSNESS). The trussness of a subgraph $H \subseteq G$ is the minimum support of edges in $H$, denoted by $\tau(H) = \min\{sup(e, H) : e \in E(H)\}$.

*Definition 5:* (EDGE TRUSSNESS). The trussness of an edge $e \in E(G)$ is the maximum trussness of $e$ in all the subgraphs, i.e., $\tau(e) = max_{H \subseteq G}\{\{\tau(H) : e \in E(H)\}$.

[2]Note that the updates could be very frequent (i.e., near real time) to meet the needs of the clients.

*Definition 6:* (VERTEX TRUSSNESS). The trussness of a vertex $v \in V(G)$ is the maximum trussness of $v$ in all the subgraphs, i.e., $\tau(v) = max_{H \subseteq G}\{\{\tau(H) : v \in V(H)\}$.

The social cohesiveness group defined by $k$-truss can be generalized using other dense subgraph definitions, such as the $k$-core group, clique [39] and $k$-edge-connected component community.

### A. Problem formulation

In this section, we formally define the CGSGM query.

**Problem Statement.** *Given a social graph G = (V, E), where each vertex $v \in V$ is a candidate attendee whose location at time instance $t_i$ (where $t_i \in [t_1, t_2]$) and any two mutually acquainted vertices u and v are connected by an edge $e_{u,v}$. A continuous geo-social group monitoring query CGSGM over moving users $\langle d, p, q_l, k\rangle$ continuously monitors all the k-truss groups of size p from G between time instance $t_1$ and time instance $t_2$ where the distance of every result user to the query location $q_l$ is less than d, i.e., $\forall v, dis(v, q_l) \leq d$.*

*Theorem 1:* The CGSGM query is NP-hard.

*Proof*: For each time instance, we return all the $k$-truss groups with a fixed size $p$. We prove the NP-hardness of our CGSGM problem by reducing from a decision version of the well-known NP-hard Maximum Clique problem, that is, checking whether there exists a non-empty $k$-clique in graph $G(V, E)$. We construct an instance of CGSGM for graph $G(V, E)$ with moving users $\langle d, p, q_l, k\rangle$, by setting the parameters $k = p$ and $d = +\infty$ such that every user $v \in V$ satisfies $dis(v, q_l) \leq d$. A $k$-truss of size $p = k$ is a $k$-clique, because every edge $(v, u) \in E$ has exactly $k - 2$ triangles and thus each vertex $v$ has $k - 1$ neighbors. Thus, the decision problem of Maximum Clique is a Yes-instance *if and only if* the corresponding of our CGSGM is also a Yes-instance. This completes the proof.

### B. K-truss index

In this section, we first introduce a $k$-truss index from [40] by maintaining the truss information to check whether a user is qualified to be a result user by its corresponding edge's trussness.

We first apply a truss decomposition algorithm [41] on the graph $G$, which computes the trussness of each edge, denoted as $\tau(e(u, v))$. For each vertex $v \in V$, we sort its neighbors N(v) in descending order of the edge trussness $\tau(e(u, v))$ for $u \in N(v)$ and set the maximum edge trussness as the $v$'s trussness. For each distinct trussness value $k \geq 2$, we mark the position of the first vertex $u$ in the sorted adjacency list where $\tau(e(u, v)) = k$. This supports efficient retrieval of $v$'s incident edges with a certain trussness value. We also use a hash table to keep all the edges and their trussness values. This is the $k$-truss index.

## IV. BASELINE

To address the CGSGM problem, which is to find all the $k$-truss groups of size $p$ corresponding to the query position $q_l$ and query distance threshold $d$, an idea is to first obtain



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

the users inside the circle with the radius $d$ centered at $q_l$, called *monitoring circle* (denoted as $C_m$), and then perform $k$-truss decomposition (called ***KTD***) [41] to obtain the qualified users at a scheduled time instance, and finally eumerate all the possible groups of size $p$ from these qualified users to return all the $k$-truss result groups.

Before proposing the baseline, we first discuss how to compute the initial results efficiently.

### A. The initial results computation

Given a CGSGM $\langle d, q_l, k, p\rangle$ query at the initial time instance, a natural idea is to first scan all the users to obtain a set of candidate users $S_{can}$. A candidate's distance to query location $q_l$ should be no greater than $d$, while her trussness is also not less than $k$. The trussness of users may be exploited to filter unqualified users, termed as *vertex trussness filtering*, is presented in Lemma 1. To efficiently utilize this vertex trussness filtering, we should precompute the trussness for each user before the query is processed, as stored in our $k$-truss index. After that, we invoke the $k$-truss decomposition function to remove the users who can not be a member of $k$-truss. This is called *$k$-truss decomposition filtering*, as depicted in Lemma 2. Notice that KTD is implemented by revising the truss decomposition algorithm when setting the initial truss value as $k$. After obtaining the qualified users (i.e., the maximal $k$-truss), we enumerate candidate groups of size $p$ by invoking a branch and bound (BB) algorithm. The BB algorithm is to enumerate the possible groups for finding all the connected $k$-truss groups. The pseudo-code of the initial result computation algorithm is shown in Algorithm 1.

*Lemma 1:* **Vertex trussness filtering.** Given a CGSGM query $\langle d, p, q_l, k\rangle$, the current feasible solution $U_I$, a new user $u$ can be safely filtered iff its trussness is less than $k$.

*Proof*: The lemma holds since the vertices with truness $<$ $k$ do not meet the requirement of $k$-truss in Definition 3.

*Lemma 2:* **Maximal $k$-truss filtering.** Given a user $u$ and the maximal $k$-truss $MT$, $u$ is qualified as a member of $k$-truss of size $p$ ($\leq |MT|$) iff $u \in MT$. That is, if $u$ is not in $MT$, it can be safely filtered from the group combination for forming the $k$-truss.

*Proof*: The lemma holds since the vertices that are not a part of the maximal $k$-truss clearly cannot meet the structural cohesiveness requirement in Definition 3.

The main idea of the Branch and Bound (BB) algorithm is to apply a branch and bound strategy to enumerate the candidate groups with the qualified users set $MT$. Let $U_I$ denote the intermediate solution set of candidate attendees, which is initialized with $MT$ and used for the further query processing. In addition, we use $U_R$ to denote the remaining set of candidate attendees. At each iteration afterward in BB algorithm, we select a vertex from $U_R$ to $U_I$ randomly. If the size of intermediate group $U_I$ is $p$, we check whether it is a connected $k$-truss. If it is, we push it into the results $S_{kt}$. Otherwise (the size is smaller than $p$), we continuously select a vertex $U_R$ to $U_I$. When $U_R$ becomes empty, the query processing terminates.

**Algorithm 1:** Initial result computation (IRC)

**Input**: A social graph $G = (V, E)$, a query CGSGM $\langle d, q_l, k, p\rangle$, users' location array $A_{loc}$ at the current time instance

**Output**: All the connected $k$-truss groups $S_{kt}$

```
for each user u ∈ A_loc do
    if τ(u) ≥ k and dis(u, q_l) ≤ d then
        updateNeighborandCommonNeigbor(u, S_can);
        Insert u into candidate set S_can;
MT ← KTD(S_can, k);
S_kt ← BB(MT, k);
return S_kt;
```

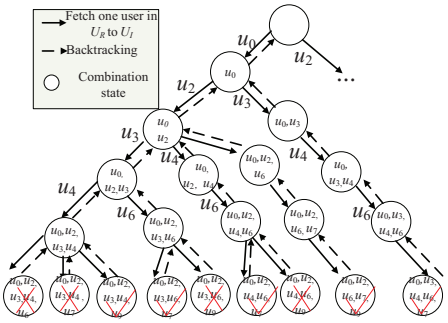



Figure 3. An example of branch and bound

*Example 1:* Recall the example in Figure 1 for illustration of the initial result computation algorithm. Given a CGSGM $\langle d,\ q_l,\ 3,\ 5\rangle$ query, at the initial time instance, the candidate users set $S_{can}$ is $\{u_0,\ u_2,\ u_3,\ u_4,\ u_6,\ u_7,\ u_9,\ u_{11}\}$. Then we invoke the KTD funtion to get the qualified users set as $\{u_0,\ u_2,\ u_3,\ u_4,\ u_6,\ u_7,\ u_9\}$. After that, the Branch and Bound algorithm is invoked to explore the candidate groups as shown in Figure 3. Firstly, $U_I$ is initialized with $\{u_0\}$. Then $u_2$ is selected to form a new branch and $U_R$ $=\{u_3,\ u_4,\ u_6,\ u_7,\ u_9\}$. By selecting a vertex from $U_R$ to $U_I$ iteratively, we first check $\{u_0, u_2, u_3, u_4, u_6\}$, $\{u_0, u_2, u_3, u_4, u_7\}$ and $\{u_0, u_2, u_3, u_4, u_9\}$ of size 5, sequentially. Now we find one connected 3-truss group $\{u_0, u_2, u_3, u_4, u_9\}$. Next, we continue to explore the branch $\{u_0, u_2, u_3, u_6\}$. Similarly, $\{u_0, u_2, u_3, u_6, u_7\}$ is checked as a 3-truss group. Then $\{u_0,\ u_2,\ u_9,\ u_6,\ u_7\}$ is also returned as a 3-truss group when exploring branch $\{u_0, u_2, u_9\}$. After that, the branches $\{u_0, u_3\}$, $\{u_2\}$ and $\{u_3\}$ are explored and the corresponding 3-trusses $\{u_0, u_3, u_4, u_6, u_7\}$ and $\{u_0, u_3, u_9, u_6, u_7\}$ are returned. At this stage, the query processing ends and the size of remain users is smaller than 5. All the connected 3-truss groups of size 5 are $\{u_0, u_2, u_3, u_4, u_9\}$, $\{u_0, u_2, u_3, u_6, u_7\}$, $\{u_0, u_2, u_6, u_7, u_9\}$, $\{u_0, u_3, u_4, u_6, u_7\}$, and $\{u_0, u_3, u_9, u_6, u_7\}$.

As shown, the branch and bound idea explores many invalid combinations, e.g., the branches $\{u_2\}$, $\{u_3\}$ and $\{u_0, u_2, u_4\}$, thus it is inefficient to answer the CGSGM query.



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

**Improvement by the CNNE strategy.** We argue that BB is not efficient to form a valid combination by fetching a user randomly. Thus, we propose a new expanding strategy, *common neighbor or neighbor expanding (CNNE)*, to quickly produce the feasible solutions, which also explores some effective pruning and filtering techniques.

For the sake of expanding a valid common neighbor, we introduce the edge trussness filtering as follows.

*Lemma 3:* **Edge Trussness filtering.** Given a CGSGM query $\langle d, p, q_l, k\rangle$, a new edge can be safely filtered iff its trussness is less than $k$.

*Proof*: The lemma holds according to the requirement of $k$-truss in Definition 3.

Assume that one common neighbor $v$ for edge $e(x,y)$ is considered for expansion. According to the edge trussness filtering, if the truss of one edge $e(v,x)$ (or $e(v,y)$) is less than $k$, such a common neighbor $v$ is not valid for expansion.

**The CNNE strategy.** To quickly form a feasible truss group, it is a natural idea to use one edge's common neighbor to expand the current group so that this can increase this edge's support. For introducing this idea in detail, let's reconsider Example 1. Similarly, we first compute the maximum 3-truss set $S_{can} = \{u_0, u_2, u_3, u_4, u_6, u_7\ u_9\}$ for the initial time instance. After that, group combination exploration is performed. We also choose the user $u_0$ in the running example. Then one $u_0$'s neighbor $u_2$ whose edge trussness is greater than $k$ is selected according to the property of a connected $k$-truss. After that, we may select the common neighbor of edge $e(u_0, u_2)$ in the intermediate set $U_I$ to expand $U_I$. Continue with these steps until all the possible combinations are enumerated. However, two key questions may arise during expanding $U_I$:

- Which edge should be first selected to expand its common neighbors when there are many edges in $U_I$?
- Is it correct to always select a common neighbor to expand $U_I$?

To answer these two questions, let's show an example in Figure 1. Give a CGSGM query $\langle d, q_l, 3, 5\rangle$ at time instance $t_1$. For this query, we first select $u_0$ and its neighbor, e.g., $u_2$. At this time, for branch ($u_0$, $u_3$) to become a 3-truss, the support of edge $e(u_0,\ u_3)$ should be at least 1. Thus, at least one common neighbor of $e(u_0,\ u_3)$ should be added to the group to support this edge. Since the common neighbors of edge $e(u_0,u_2)$ are $u_3$ and $u_9$. At this moment, we have two choices to expand $U_I$=$\{u_0, u_3\}$. If we select $u_3$, then $U_I$ becomes $\{u_0, u_2, u_3\}$, which has three edges. Notice that the support of all these three edges is not less than 1. Now, we should decide how to expand $U_I$ to be a 3-truss of size 5. If we always continue to expand one edge's common neighbor, e.g., $u_9, u_4$, we can find only one feasible group $\{u_0, u_2, u_3, u_4, u_9\}$. However, group $\{u_0, u_2, u_3, u_7, u_6\}$ is missing in the final results since both $u_7$ and $u_6$ are not the common neighbor of any edge in $\{u_0, u_2, u_3\}$.

From the above example, we can see that it is not a good idea to always select a common neighbor to expand the intermediate group. Thus, we develop Lemma 4, for one edge's support w.r.t the current $U_I$.

*Lemma 4:* Given an intermediate set $U_I$, if the support of an edge $e$, denoted as sup($e$), is less than $k$-2, $U_I$ may be expanded as a feasible solution only if there are at least $k$-2-$sup(e)$ common neighbors to be added to $U_I$; Otherwise, this branch can be safely pruned.

*Proof*: It is easy to be proved according to the support condition of $k$-truss in Definition 3.

To answer these two questions, we observe that there exist two cases with respect to edges' support for an intermediate solution set $U_I$ where $|U_I|$ is smaller than $p$.

- **Case 1**. All edge's support is not less than $k-2$.
- **Case 2**. There exist at least one edge with support less than $k-2$.

Accordingly, in order to efficiently expand one branch, we propose a new expanding scheme, called *edge support based decision* (ESD) scheme, which mainly chooses the common neighbors of the edge whose support is less than $k$-2 or the neighbors of users when all the edge's support is equal or larger than $k$-2 w.r.t $|U_I|$, to expand the intermediate solution set $U_I$ iteratively. Thus, the above two cases are addressed as follows.

(i) For the first case, we have two situations: (1) if the size of $U_I$ is $p$-1, we follow Lemma 5 below; (2) otherwise, we should fetch the neighbors of users instead of the common neighbors to expand current $U_I$.

*Lemma 5:* Given an intermediate set $U_I$ of size $p$-1, the support of all the edges is not less than $k$-2. The seed $U_I$ can expand as a feasible solution only if there must be one common neighbor of one edge for adding to $U_I$.

*Proof*: The lemma holds according to the support condition of $k$-truss in Definition 3.

However, for the second situation, many neighbors of $U_I$ should be explored to expand the current group and a lot of branches are produced. However, some branches are invalid. To reduce these invalid branches, we propose Lemma 6 to check whether a given neighbor is valid to form a feasible branch.

*Lemma 6:* Consider an intermediate set $U_I$ where the support of all the edges is not less than $k$-2 and one new user $v$ is one neighbor of $U_I$. $U_I \cup \{v\}$ is qualified as a feasible solution only if there exists one edge $e(v,x)$ ($x \in U_I$) whose trussness is not less than $k$.

In Figure 1, give a CGSGM query $\langle d, q_l, 3, 4\rangle$ and current $U_I$= $\{u_0, u_3, u_4\}$. Since the size of $U_I$ is 3, if we expect $\{u_0, u_3, u_4\}$ to become a 3-truss of size 4, we can't add one neighbor of nodes, e.g., $u_7$, $u_8$, according to Lemma 3. We should add one common neighbor of edge $e(u_0,u_3)$, e.g., $u_2$, $u_9$ and $u_1$, to be a 3-truss of size 4, while there is no commnon neighbor for edges $e(u_0,u_4)$ and $e(u_3,u_4)$.

(ii) With regard to the second case, since there may exist several edges whose support is smaller than $k$-2, it is necessary to design a scheme to decide an appropriate edge $e$ for expanding its common neighbors.



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

**Deciding an appropriate edge.** For efficient expanding one branch, it is involved in two aspects: (1) if one branch can not produce a good feasible solution, it is necessary to terminate this branch quickly; and (2) otherwise, it is important to produce the good feasible solution earlier.

With regard to the first aspect (i.e., terminate this branch quickly), the size of the valid common neighbors of the edge $e$, denoted as $|CN(e)|$, is essential for expanding one branch. That is, if that size is smaller, the number of combinations to be examined is less. Thus if we select the edge whose common neighbor size is small to expand current branch, we can reduce the number of combinations exploration. For example, given the current $U_I$= $\{u_0, u_8, u_9\}$, the common neighbors of $e(u_0, u_9)$ are $\{u_5, u_1, u_2, u_3\}$, while the common neighbors of $e(u_8, u_9)$ only contains $u_5$. Thus for this, we choose to use $u_5$ for edge $e(u_8, u_9)$ as the expanding user and quickly determine $\{q, u_8, u_9, u_5\}$ as a feasible group by only checking one combination. In contrast, if we decide $e(u_0, u_9)$ as the expanding edge, the number of combinations checking is 4, that requires to check more combinations than using edge $e(u_8, u_9)$.

Moreover, it is vital for considering the *remaining support* of one edge as defined in Definition 7 for the first aspect.

*Definition 7:* (REMAINING SUPPORT.) Given a sub-graph $G'$ and social constraint $c$, we say the remaining support of edge $e$, denoted as $\widehat{sup(e)}$, is $k-2-sup'_G(e)$, where $sup'_G(e)$ is the support of $e$ with respect to $G'$ and is smaller than $k$-2. Notice that if the support of $e$ with respect to $G'$ is not less than $k$-2, then $\widehat{sup(e)}$=0.

The idea behind *remaining support* is that we need to add $\widehat{sup(e)}$ commnon neighbors to support this edge to be a $k$-truss. Thus, we have a lemma, denoted as *support pruning*.

*Lemma 7:* Given an intermediate set $U_I$, if the size of one edge's common neighbors is smaller than $\widehat{sup(e)}$, $U_I$ can be safely pruned.

*Proof*: The proof is trivial according to the support condition of $k$-truss in Definition 3.

If the remaining support of one edge in $U_I$ is smaller, we require to expand less users for supporting such an edge. Thus, this can reduce the number of combinations examination. Therefore, according to the size of valid common neighbors ($|CN(e)|$) and remaining support ($\widehat{sup(e)}$), we give priority to selecting the edge whose $\widehat{sup(e)}$ is smaller than $|CN(e)|$ for expansion. This is because $k$ in a query is always very small, while the size of common neighbors may be very large.

From the above CNNE strategy, however, there is one case that current branch combining the common neighbor may not produce one feasible solution. For example, consider a CGSGM query $\langle d, q_l, 4, 4\rangle$ and the current intermediate result $U_I = \{u_0, u_2, u_3\}$ generated by the CNNE strategy. If we choose to use the common neighbor $u_4$ to expand, we can not produce a feasible solution finally. The main reason is that the diameter of $\{u_0, u_2, u_3, u_4\}$ is 2, which can not satisfy the diameter property of a connected $k$-truss below. Thus, we also propose a more effective pruning strategy, as shown in Lemma 8, by exploiting this diameter property.

*Property 1:* The structural diameter of a connected $k$-truss with $n$ vertices is no more than $\lfloor\frac{2*n-2}{k}\rfloor$ [32].

*Lemma 8:* Given a social graph $G$, and an intermediate set $U_I \subseteq G$ of size $n$, for vertex $v$, $u \in U_I$, $U_I$ becomes a connected $k$-truss only if $\text{gd}(v, u, G) \leq \lfloor\frac{2*n-2}{k}\rfloor$, where $\text{gd}(v, u, G)$ denotes the graph distance between $u$ and $v$ in $G$.

*Proof*: According to Property 1, it is easy to deduce that Lemma 8 is correct.

For obtaining the common neighbors for one edge $e(x, y)$, we have two ways: (1) online computing the common neighbors, which checks the neighbors of one vertex $x$ whose trussness is larger than $k$ and also check the neighbor with $y$ is also edge connected and the corresponding trussness is also larger than $k$; (2) offline precomputing the common neighbors with the minimal truss since we do not know the $k$ value before the query comes. Moreover, during the query processing, we just choose the common neighbors whose trussness is larger than $k$ online.

This common neighbor expanding is mainly dependent on the support of the edge. If the support does not satisfy, we continue to add the nodes to satisfy the support of this edge.

Different from the branch and bound by randomly fetching users to form the combinations, the common neighbor or neighbor expanding (CNNE) strategy mainly explores the group combinations by expanding the common neighbors of edges or neighbors of users in $U_I$ in each iteration with the help of ESD scheme and utilizing effective filtering strategies (i.e., Lemma 4, 5 and 6). The pseudo-code is shown in Algorithm 5. Similar to the BB strategy, our CNNE mainly expands the branch by fetching the neighbors or the common neighbors from the remaining users set $U_R$ to form the intermediate solution set $U_I$. $U_R$ is initialized with the maximal k-truss $MT$. For the expanding process, we first use the ESD scheme to just check whether all the edges' support in $U_I$ is not less than $k$-2 (Line 5). In the sequel, CNNE expands the current $U_I$ by selecting one user from remaining users $U_R$ according to the $e$ status (Lines 9-15). For each user in $U_R$, we check whether it is common neighbor or neighbor according to whether $e$ exists and continue to expand the branch. If for current $|U_I|$, there is no valid user for expanding, i.e., $isExpanding = false$, we just terminate this branch.

Figure 4 shows an example for illustrating the CNNE strategy. Give a CGSGM query$\langle d, q_l, 3, 5\rangle$ for time instance $t_1$. According to the order in $U_R$ =$\{u_0, u_2, u_3, u_4, u_6, u_7, u_9\}$ initialized with the maximal 3-truss set, $u_0$ and $u_2$ are first fetched to form current $U_I$. By selecting a vertex from $U_R$ to combine $\{u_2, u_0\}$ according to the CNNE strategy, we first consider the common neighbor $u_3$ of $e(u_0,u_2)$ and the intermediate solution $U_I$ becomes $\{u_2, u_0, u_3\}$. At this time, the support of all the edges (i.e., $e(u_0, u_2)$, $e(u_0, u_3)$ and $e(u_3, u_2)$ ) is 1 and satisfy the truss condition. Thus, we need to get one neighbor of $U_I$ to expand and $u_4$ is selected to form $U_I$ =$\{u_0, u_2, u_3, u_4\}$. Also for current $U_I$, the support of all edges is 1. According to Lemma 5, since the size of



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

**Algorithm 2:** The CNNE Function

**Input**: Intermediate solution set $U_I$, remaining users set $U_R$, group size $p$ and $k$
**Output**: CGSGM result groups $RG$

```
if |U_I| = p then
    if U_I is a k-truss then
        return RG;
else
    e ← ESD(U_I);/*Find the appropriate edge; */
    while U_R is not empty do
        for each user u_i ∈ U_R do
            if (e = ∅ and u_i is one neighbor of users in U_I) or (e ≠ ∅ and u_i is one common neighbor of e) then
                isExpanding ←true;
                tempU_I ← U_I;
                tempU_I.push_back(u_i);
                delete u_i from U_R;
                CNNE(newU_I, U_R);
        if isExpanding = false then
            return ∅;
```

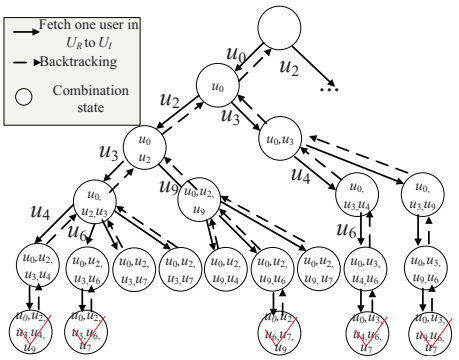



Figure 4. An example of CNNE

$U_I$ is 4 (i.e., $p$-1), we should combine a common neighbor of one edge and only $u_9$ is the common neighbor of the edges. Thus $u_9$ is examined to combine $U_I$ which becomes $\{u_0, u_2, u_3, u_4\ u_9\}$. Now, the size of $U_I$ is 5, so we check whether $U_I$ is a connected 3-truss and find $\{u_0, u_2, u_3, u_4\ u_9\}$ as a result group. After that, we go back to branch $\{u_0, u_2, u_3\}$ and consider another neighbor $u_6$ and $U_I$ becomes $\{u_0, u_2, u_3, u_6\ \}$. Now, only the support of edge $e(u_0, u_6)$ is 0, so we add the commnon neighbor of this edge to support this edge. Accordingly, $u_7$ is inserted into $U_I$. Similarly, another result group $\{u_0, u_2, u_3, u_6, u_7\}$ is found. In the sequel, we continue to combine neighbor $u_7$ and $u_9$ for current $U_I$= $\{u_0, u_2, u_3\}$, but we find that these two neighbors can not produce a valid branch. Continue with the above steps using CNNE until $U_R$ becomes empty and the algorithm safely terminates.

### B. The baseline algortihm

For each time instance, we compute the CGSGM query from stratch using the initial result computation algorithms. Based on the above two initial result computation algorithms, our baseline has two algorithms, namely Baseline-BB and Baseline-CNNE.

In fact, during the query processing, the baseline recomputes the support of all users from scratch when users are moving into or move out of the monitor circle, i.e., the circle centered at $q_l$ of radius $d$, which do not exploit the intermediate results to help further query processing. Thus, a natural idea is to strike for an efficient way to maintain some intermediate results to accelerate the following query processing for the new time instance. Motivated by this, our idea is to design an incremental algorithm by maintaining the neighbors of users and edge's common neighbors inside the monitor circle in each time instance.

## V. INCREMENTAL MONITORING ALGORITHM

In this section, we propose an incremental monitoring algorithm, namely IMA, consisting of initial result computation and multiple users update. For the initial result computation, we employ the baseline with CNNE to compute the initial result with the candidate users $S_{can}$, at the same time we maintain the support, common neighbors and neighbors under the current users inside the monitoring circle $C_m$. Thus, our main task is to efficiently process the users updates.

In the monitor framework, only the users are moving. We assume that the query position is static. For simplicity, we first analyze the case where only one user moves/updates. Next, we discuss the general cases where multiple users update/move simultaneously.

### A. Processing of one user update

For one user update, there are two scenarios with respect to the monitor circle $C_m$, i.e., the user is outside of the circle and inside the circle. For a user, the user is already a result user or an non-result user. A user update contains the user id $u.id$, and its old and new coordinates. Thus, there are four cases.

(1) An inside user moves out of (leave) the circle $C_m$. For this case, it invalidates some result groups and thus does not produce any new connected $k$-truss. If the trussness of this user is smaller than $k$, we need do nothing. Otherwise, we need to remove this user from $S_{can}$ and update the support, the common neighbors to the edges whose trussness under $G$ is not less than $k$, neighbors of the users in $S_{can}$.

(2) An outside user still moves outside the monitoring circle. This case does not alter any result.

(3) An inside user is still moving into the circle $C_m$. For this case, the query result does not change and we need not do anything.

(4) An outside user moves into the circle $C_m$. For this case, new $k$-truss groups may be generated due to the new users adding if the trussness of this user is not less than $k$. To efficiently decide whether the new user can produce a new



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

**Algorithm 3:** Delete one user (DU)

**Input**: The previous users set $S_{can}$, a query CGSGM $\langle d, q_l, k, p\rangle$, and one leaving user $u$

1 **if** $\tau(u) < k$ **then**
2 return;//This user never changes the results;
3 US($sup, u$);
4 Update the neighbors of $w$ and common neighbors for each $w \in (N_G(u) \cap S_{can})$ with $u$;

$k$-truss group with the original users, we propose an algorithm for adding one new user. When a new user $u$ is moving into the circle, first we need to check its truss value is no less than $k$. If it is not, this user can not produce a connected $k$-truss and we terminate the query processing. Otherwise, we need to update the support, common neighbors and neighbors of $u$. If the maximal support among these egdes involved in u is less than $k$-2, adding this user also does not produce a new $k$-truss. After that, we invoke the $k$-truss decomposition function to get the maximum $k$-truss and check whether the maximum $k$-truss contains $u$. If it does not, adding $u$ does not produce a new connnected $k$-truss and we terminate the query process. Otherwise, we invoke CNNE by initiating $U_I$ with $u$ to get all the $k$-truss groups including $u$ of size $p$ from the maximum $k$-truss. Finally, we return all the result groups in $S_{kt}$.

**Algorithm 4:** Add new user (AU)

**Input**: The current users set $S_{can}$, a query CGSGM $\langle d, q_l, k, p\rangle$, and one adding user $u$

**Output**: The new connnected $k$-truss groups

1 **if** $\tau(u) < k$ **then**
2 return;//This user never changes the results;
3 UpdateSupport($sup, u$);
4 Update the neighbors of $w$ and corresponding common neighbors for each $w \in N_G(u) \cap S_{can}$ with $u$;
5 **if** $max\{sup(v, u)\}|v \in V) < k - 2$ **then**
6 return $\varnothing$;
7 $MK \leftarrow$ KTD($u, S_{can}$);
8 **if** $MK$ does not contain $u$ **then**
9 return $\varnothing$;
10 $S_{kt} \leftarrow$ CNNE($u$, $MK$, $k$);
11 return $S_{kt}$

The basic idea of update support (US) function is to first check whether this new user is deleted from or added to the monitoring circle. If this user is deleted from $S_{can}$, we check each edge $e(x, y)$ to see whether it may form a triangle with $u$. If it is, the support of $e(x, y)$ minus one and other two edges are deleted. While for adding user $u$, we also check each edge $e(x, y)$ to see whether it may form a triangle with $u$. If it is, each support of three edges is added by one.

### *B. Processing of multiple-users update*

So far we have considered each type of updates individually. In this section we deal with concurrency issues that may arise in the general case, where updates of all four types arrive simultaneously at the system for multiple users. Our aim is to process the updates in a batch to save as much computation as possible.

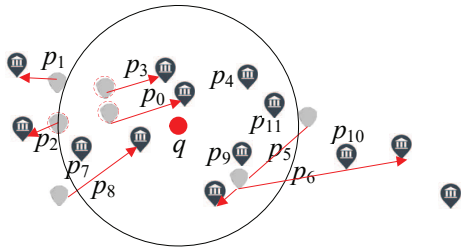



Figure 5. An example of multiple-users update

A direct idea is processing the updates one by one in certain order. Consider the example in Figure 5 to show multiple users' updates for query $\langle d,\ q_l,\ 3, 5\rangle$. The previous candidate users set $S_{can}$ is $\{u_0,\ u_2,\ u_3,\ u_4,\ u_6,\ u_7, u_9, u_{11}\}$. For those updates with red arrows shown in the figure, e.g., $u_8$ moves into the circle and $u_2$ moves out of (leave) the circle, if we process the update of $u_8$ first (case 4), a new user $u_8$ is added and the AU algorithm is invoked to check whether there is a new connected 3-truss to be generated. Accordingly, we find a new connected 3-truss group including $u_2$. However, this new group is not a valid 3-truss, since $u_2$ moves outside the circle at the same time. On the other hand, if we process the updates of $u_2$ and $u_6$ first by deleting these two users, then process the update of $u_8$. This processing order produces two new connected 3-trusses $\{u_0, u_3, u_7, u_8, u_9,\}$ and $\{u_0, u_3, u_4, u_8, u_9\}$, which are valid. Although a new user $u_5$ is also moving into the circle, we can find the new 3-truss group by including $u_8$ and $u_5$ after processing $u_8$. Thus, selecting a good order to process the updates is very essential for the query processing.

Motivated by the above example, we decide the order to process location updates based on whether the user updates affect the group generation. One feasible idea is processing the leaving users first and then processing the adding users. Based on this, we propose a multiple user update algorithm.

Our idea is processing the updates simultaneously. We first check which case each moving user belongs to. According to the four cases, to delete a user who is in $S_{can}$, we remove this user from $S_{can}$ and update the supports of involved edges, corresponding degrees and neighbors. With regard to adding user whose truss value is greater than $k$ and who is also not in $S_{can}$, we insert it into adding user set $S_a$. After simply processing each leaving user, we invoke the AU function to process the users one by one in $S_a$.

An example of processing the multiple user updates for query $\langle d, q_l, 3, 5\rangle$ is shown in Figure 5. The previous candidate users set $S_{can}$ is $\{u_0,\ u_2,\ u_3,\ u_4,\ u_6\ u_7, u_9, u_{11}\}$. Then we process each moving user one by one. As the updates of $u_0$ and $u_3$ do not alter anything, we skip those updates. For the updates of $u_2$ and $u_6$, since they move out of monitoring



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

**Algorithm 5:** The multiple users update algorithm

**Input**: A social graph $G = (V, E)$, a query CGSGM $\langle d, q_l, k, p\rangle$, and previous candidate users set $S_{can}$
**Output**: The new $k$-truss groups

```
for each moving user u do
    if u ∈ S_can then
        if dis(u, q_l) > d then
            DU(S_can, u);
    else
        if dis(u, q_l) ≤ d and τ(u) >= k then
            insert u into S_a;
for each user u ∈ S_a do
    S_kt ← AU(u, S_can);
    insert u into S_can;
return S_kt;
```

circle, we need to delete them and update the corresponding support, neighbors and the common neighbors. While for $u_5$ and $u_8$, they are added to $S_{can}$. Thus, we need to combine these two users with the users in $S_{can}$. We first combine $u_5$ with $S_{can}$ by invoking the AU function. For this, we get three result groups $\{u_0, u_3, u_4, u_7, u_5\}$, $\{u_0, u_3, u_4, u_9, u_5\}$ and $\{u_0, u_3, u_9, u_7, u_5\}$ of size 5. After that, $u_8$ is explored, we also obtain five 3-truss groups $\{u_0, u_3, u_4, u_5, u_8\}$, $\{u_0, u_3, u_4, u_7, u_8\}$, $\{u_0, u_3, u_9, u_5, u_8\}$, $\{u_0, u_3, u_9, u_7, u_8\}$ and $\{u_0, u_5, u_7, u_9, u_8\}$ and the query processing terminates.

**Complexity Analysis.** We analyze the time complexity of the two algorithms proposed for a CGSGM query $\langle d, q_l, k, p\rangle$ in this paper. Let $|V|$ be the number of users, i.e., the size of vertices. The time complexity of these algorithms can be derived as follows. We first analyze the time complexity of initial result computation. For Baseline-BB, it first scan all the users and obtain the candidate users $S_{can}$, which it takes O($|V|$) time. After that, the KTD function with $S_{can}$ is invoked to find the maximal $k$-truss, which takes O($n_c \times \log m_e$), where $n_c$ is the number of users in $S_{can}$ and $m_e$ is the size of corresponding edges. Then for the remaining users in the maximal $k$-truss, it takes O($n_{mt}^p$) to enumerate the group combinations by invoking the BB function to find all the $k$-truss groups, where $n_{mt}$ is the number of the users in the maximal $k$-truss. In total, the time complexity of the Baseline-BB algorithm at each time instance is O($|V|$+ $n_c \times \log m_e$ +$n_{mt}^p$). While for the Baseline-CNNE, the difference is to enumerate the group combinations by invoking the CNNE function after obtaining the maximal $k$-truss. For invoking CNNE function with the maximal $k$-truss, it takes O($n_{mt} \times n1_{max}^{p-1}$) where $n1_{max}$ is the maximal number of the common neighbors of one edge whose trussness is not less than $k$. Thus, the time complexity of the Baseline-CNNE algorithm is O($|V|$+ $n_c \times \log m_e$ +$n_{mt} \times n1_{max}^{p-1}$).

For IMA, the initial result computation is the same as the baseline. At each new time instance, we assume that there are $n_d$ users to be deleted from the previous candidate users set $S_{can}$. For each leaving user $u$, it takes O($n_e$) to update the support and common neighbors for $n_e$ edges which involved with $u$. Thus, it takes O($n_d \times n_e$) to process those deleting users. In addition, we assume that there are $n_a$ users to be added into $S_{can}$. For each adding user $v$, it takes O($n_{c1} \times \log m_{e1}$) to do the truss decomposition, where $n_{c1}$ is the number of the users that have deleted those valid users and added $v$, and $m_{e1}$ is the size of corresponding edges. Thus, $n_{c1}$ is smaller than $n_c$ in baseline. Then it takes O($n2_{max}^{p-1}$) to invoke CNNE to explore the possible combinations with $v$, where $n2_{max}$ is the size of users after truss decomposition. Thus, the time complexity of the IMA algorithm is O( $n_d \times n_e$ + $n_a \times$ ($n_{c1} \times \log m_{e1}$+ $n2_{max}^{p-1}$)).

**Extension to handle multiple queries.** In addition, we extend IMA to handle multiple queries simultaneously, aiming to achieve high throughput. To do so, we build a grid index to index the registered queries, where each grid cell is associated with all the queries whose monitoring circles intersect with the cell. Upon a user location update, the affected queries can be quickly identified through the grid index and re-evaluated using the incremental update algorithm. Note that the re-evaluations of different queries can be performed in parallel since their results are independent with each other.

## VI. EXPERIMENTS

In this section, we evaluate the performance of the proposed algorithms for the CGSGM query. All the algorithms are implemented in C++, while the experiments are conducted on an Intel Core i5 2.3 GHz PC with 16GB RAM.

We conduct experiments on three real datasets. We simulate the user movements based on the following datasets.

- ***Gowalla***. It consists of 196585 nodes and 138337 edges for the social network. The percentage of moving users is 4.89% per minute.[3]
- ***Brightkite***. It contains 58228 users and 214078 edges. The percentage of moving users is 6.86% per minute.[3]
- ***Foursquare***. It consists of 2146576 nodes and 8919127 edges for the social network. The percentage of moving users is 5.46% per minute.[3]

Table I
PARAMETER RANGES AND DEFAULTS VALUES

| Parameters | Range |
|---|---|
| ♯ of selected users ($p$) | 3, **4**, 5, 6, 7, 8 |
| ♯ of social constraint ($k$) | 3, **4**, 5, 6, 7 |
| distance constraint ($d$) | 0.0125, **0.025**, 0.05, 0.1, 0.2 |
| interval of time instances [4] (minutes) | 0.5, **1**, 2, 4, 8 |

We conduct a performance evaluation on the efficiency of the CGSGM algorithms – we compare the latency of the proposed CGSGM algorithms under various parameters (summarized in Table I, numbers in bold are the default settings). We measure the average latency at the time of user updates (i.e., total time/number of time-points) as the performance metric corresponding to five different parameters: (a) query distance $d$; (b) group size $p$; (c) social constraint $k$; (d) interval of time instances and (e) number of time instances.

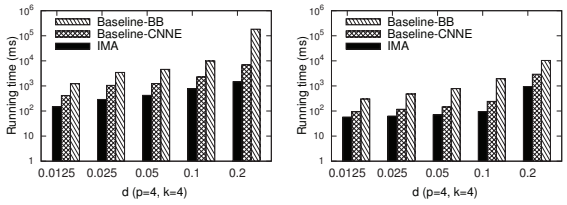
[3]https://snap.stanford.edu/data/index.html.



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

We randomly generate three groups of CGSGM queries corresponding to each dataset where each group consists of 100 CGSGM queries. In each experiment, we test one parameter at a time (by fixing other parameters at their default values). The location update is reported to the server once per time instance. The average moving speed is about 42km per hour. The distance of each location update varies from 0 to 1.29km. The reported experimental results are obtained by averaging the processing time of queries.

### A. *Efficiency of CGSGM Algorithms*




(a) Gowalla dataset (b) Brightkite dataset

Figure 6. Performance vs. query distance $d$

In this section, we compare the efficiency of the proposed IMA algorithm with two baseline algorithms, Baseline-BB and Baseline-CNNE. Accordingly, we test the three CGSGM algorithms using different datasets as described above. We take an offline approach to obtain the common neighbors for each edge in our implementation. The offline computation is not long. For example, it takes about 12.7 sec for the Gowalla dataset.

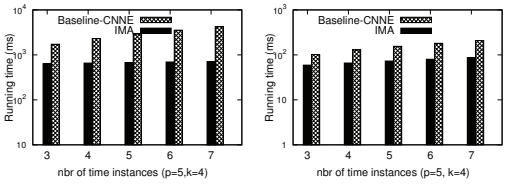
**Effect of distance threshold**. We first test the performance of the three algorithms by varying query distance $d$. Figure 6 shows the results of the three algorithms on Gowalla and Brightkite dataset. As expected, when increasing the query distance on Gowalla dataset, the query time becomes longer. This is because there will be more users to be examined as the truss result when increasing the query range distance. In addition, our IMA outperforms the two baseline algorithms a lot because our IMA incrementally maintains some intermediate results, e.g., trussness and common neighbors, which can save some query process cost. The similar results are displayed on Brightkite dataset. Since the query time of Baseline-BB is very long, we will not show the time of Baseline-BB in the remaining query results.

**Effect of $p$ value**. We then compare the average running time of IMA and Baseline-CNNE for processing CGSGM queries by increasing the group size $p$. Figure 7(a) shows the result on the Gowlla dataset. As shown, IMA performs much better than Baseline-CNNE because IMA maintains the intermediate results, e.g., the support and the common neighbor at each time instance to accelerate the further query processing, which reduces the time of $k$-truss decomposition and the number of combinations examination. When the $p$ value increases, the query time becomes longer, as more users need to be examined and the number of combinations becomes larger. Note that IMA scales better than Baseline-CNNE by increasing the group size. The results on other two datasets are similar to that on the Gowalla dataset.

**Effect of social constraint** $k$. Figure 8 compares the performance of two CGSGM algorithms by varying $k$. For the Gowalla dataset, IMA outperforms Baseline-CNNE significantly under various $k$ values because IMA incrementally examines the group combinations which avoids some redundant group combinations compared to Baseline-CNN. Moreover, the processing time of both two algorithms decreases as the $k$ value increases. This is because there are less valid users with trussness $\geq k$ when $k$ becomes larger. The results on the Brightkite dataset in Figure 8(b) and Foursquare dataset in Figure 8(c) show the similar trend with the result on the Gowalla dataset.

**Effect of the interval of time instances**. In addition, we compare the two CGSGM algorithms by varying the interval of time instances and show the results in Figure 9. The number of user updates in different duration is different. If the interval is longer, it is involved in more user updates. For the *Gowalla* dataset, the results in Figure 9(a) shows the superiority of IMA over the Baseline-CNN algorithm. The search time of IMA only takes less than 1000 ms, but the Baseline-CNN algorithm takes more than 3000 ms, because IMA maintains the support, common neighbors effectively to help accelerate the query processing. Moreover, when the query duration of each time instance becomes longer, which means more user updates, the query time of IMA and Baseline-CNN becomes longer. Similar results on Brightkite and Foursquare dataset are observed in Figures 9(b) and 9(c).

**Effect of number of time instances**. Finally, we test the two CGSGM algorithms by varying the number of time instances. As shown in Figure 10(a) on Gowalla dataset, the time of both two algorithms grows linearly when increasing the number of time instances. Also, the results show that IMA is faster than Baseline-CNNE. The results of Brightkite data are similar to that on Gowalla dataset.



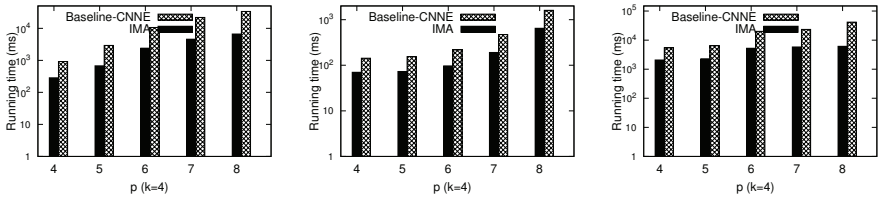

(a) Gowalla dataset (b) Brightkite dataset

Figure 10. Performance vs. number of time instances

**Efficiency of IMA for handling multiple queries.** We test the throughput for handing multiple queries simultaneously. In the experiments, we set the maximum thread number to 16 for the multi-thread processing. We perform a stress testing that continuously increases the number of queries for execution until the total execution time spent over all the queries exceeds one minute. We report the throughput of IMA by varying $p$ from 4 to 8. Figure 11(a) shows that IMA processes more than

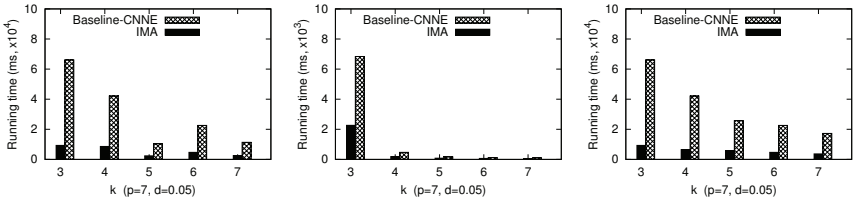


Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.



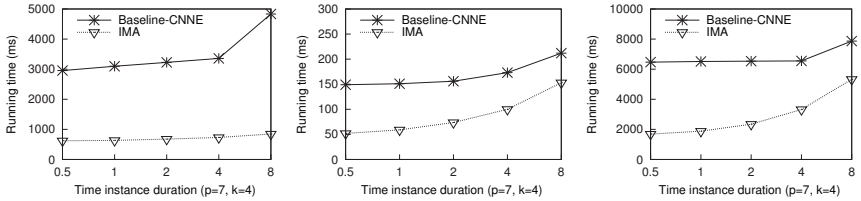

(a) Gowalla dataset (b) Brightkite dataset (c) Foursquare dataset

Figure 7. Performance vs. query group size $p$




(a) Gowalla dataset (b) Brightkite dataset (c) Foursquare dataset

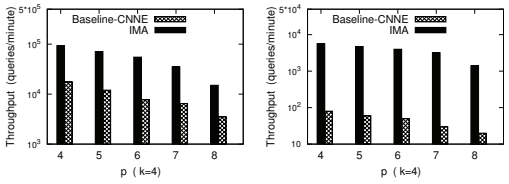
Figure 8. Performance vs. social constraint $k$




(a) Gowalla dataset (b) Brightkite dataset (c) Foursquare dataset

Figure 9. Performance vs. interval of time instances

10,000 queries per minute on the Brightkite dataset for all the $p$ values. In contrast, Baseline-CNNE processes more than 3,000 queries only. When $p$ becomes larger, the throughput becomes lower as expected. On the Foursquare dataset (see Figure 11(b)), IMA processes at least 1,000 queries per minute for all the $p$ values, while Baseline-CNNE processes less than 100 queries. This is because Baseline-CNNE requires recomputing all the information from the scratch when processing the user updates in a new time instance.




(a) Brightkite dataset (b) Foursquare dataset

Figure 11. Performance of handling multiple queries vs. $p$

## VII. CONCLUSION

In this paper, we formulate a new query, namely, *continuous geo-social groups* monitoring (*CGSGM*) over moving users, aiming to identifying continuously all the social groups appearing within a monitored geographical area. Based on different expanding strategies, i.e., BB and CNNE, we first propose two baseline algorithms to tackle the CGSGM problem. To address the shortcomings of the baseline algorithms, we then propose an incremental algrithm, namely IMA, which maintains the support, common neighbors and neighbors of the current users when exploring possible $k$-truss groups at the time of query processing for future query processing upon new user position updates. At last, we conduct a comprehensive performance evaluation using three real datasets to validate our ideas. The results show that IMA outperforms the two baseline algorithms significantly.

## ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (U1911203, 61902438, 61902439),Key-Area Research and Development Program of Guangdong Province (2020B0101100001), Guangdong Basic and Applied Basic Research Foundation (2019B1515130001), Natural Science Foundation (2019A1515011704,2019A1515011159), National Science Foundation of USA under Grant No. IIS-1717084, and Hong Kong RGC Grant 12202221, 12200021, and 12201018. Wei Liu is the corresponding author.



Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.

## REFERENCES


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

[2] Q. Zhu, H. Hu, C. Xu, J. Xu, and W. Lee, "Geo-social group queries with minimum acquaintance constraint," *VLDBJ*, 2014.

[3] Y. Li, R. Chen, L. Chen, and J. Xu, "Towards social-aware ridesharing group query services," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 646–659, 2015.

[4] C. Shen, D. Yang, L. Huang, W. Lee, and M. Chen, "Socio-spatial group queries for impromptu activity planning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 196–210, 2016.

[5] Y. Li, R. Chen, J. Xu, Q. Huang, H. Hu, and B. Choi, "Geo-social k-cover group queries for collaborative spatial computing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 10, pp. 2729–2742, 2015.

[6] B. Ghosh, M. E. Ali, F. M. Choudhury, S. H. Apon, T. Sellis, and J. Li, "The flexible socio spatial group queries," vol. 12, no. 2, pp. 99–111, 2018.

[7] T. Imielinski and B. R. Badrinath, "Querying in highly mobile distributed environments," in *18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings*, 1992, pp. 41–52.

[8] J. Qi, R. Zhang, C. S. Jensen, K. Ramamohanarao, and J. He, "Continuous spatial query processing: A survey of safe region based techniques," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 64:1–64:39, 2018.

[9] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of moving object trajectories," in *VLDB 2000*, 2000, pp. 395–406.

[10] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. López, "Indexing the positions of continuously moving objects," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, 2000, pp. 331–342.

[11] B. Gedik and L. Liu, "Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system," in *EDBT 2004*, 2004, pp. 67–87.

[12] D. V. Kalashnikov, S. Prabhakar, S. E. Hambrusch, and W. G. Aref, "Efficient evaluation of continuous range queries on moving objects," in *DEXA 2002*, 2002, pp. 731–740.

[13] G. S. Iwerks, H. Samet, and K. P. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *VLDB 2003*, 2003, pp. 512–523.

[14] M. F. Mokbel and W. G. Aref, "SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams," *VLDB J.*, vol. 17, no. 5, pp. 971–995, 2008.

[15] J. Lee, S. Kang, Y. Lee, S. J. Lee, and J. Song, "Bmq-processor: A high-performance border-crossing event detection framework for large-scale monitoring applications," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 2, pp. 234–252, 2009.

[16] M. E. Ali, E. Tanin, R. Zhang, and L. Kulik, "A motion-aware approach for efficient evaluation of continuous queries on 3d object databases," *VLDB J.*, vol. 19, no. 5, pp. 603–632, 2010.

[17] W. Huang, G. Li, K. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," in *CIKM 12*, 2012, pp. 932–941.

[18] B. Gedik and L. Liu, "Mobieyes: A distributed location monitoring service using moving location queries," *IEEE Trans. Mob. Comput.*, vol. 5, no. 10, pp. 1384–1402, 2006.

[19] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *ICDE 2011*, 2011, pp. 541–552.

[20] C. S. Jensen, J. Kolárvr, T. B. Pedersen, and I. Timko, "Nearest neighbor queries in road networks," in *ACM-GIS 2003*. ACM, 2003, pp. 1–8.

[21] Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *ACM SIGMOD 2002*, 2002, pp. 334–345.

[22] P. K. Agarwal, L. Arge, and J. Erickson, "Indexing moving points," *J. Comput. Syst. Sci.*, vol. 66, no. 1, pp. 207–243, 2003.

[23] S. Ilarri, E. Mena, and A. Illarramendi, "Location-dependent queries in mobile contexts: Distributed processing using mobile agents," *IEEE Trans. Mob. Comput.*, vol. 5, no. 8, pp. 1029–1043, 2006.

[24] H. Wang and R. Zimmermann, "Processing of continuous location-based range queries on moving objects in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 7, pp. 1065–1078, 2011.

[25] I. Afyouni, C. Ray, S. Ilarri, and C. Claramunt, "Algorithms for continuous location-dependent and context-aware queries in indoor environments," in *SIGSPATIAL 2012*, 2012, pp. 329–338.

[26] I. Afyouni and C. Ray, "A postgresql extension for continuous path and range queries in indoor mobile environments," *Pervasive Mob. Comput.*, vol. 15, pp. 128–150, 2014.

[27] X. Xiong, M. F. Mokbel, and W. G. Aref, "SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*. IEEE Computer Society, 2005, pp. 643–654.

[28] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, 2005, pp. 634–645.

[29] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 913–924, 2013.

[30] 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.

[31] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin, "Efficient computing of radius-bounded k-cores," *ICDE*, pp. 233–244, 2018.

[32] 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.

[33] B. Ghosh, M. E. Ali, F. M. Choudhury, S. H. Apon, T. Sellis, and J. Li, "The flexible socio spatial group queries," *Proceedings of the VLDB Endowment*, vol. 12, no. 2, pp. 99–111, 2018.

[34] Q. Li, Y. Zhu, and J. X. Yu, "Skyline cohesive group queries in large road-social networks," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 397–408.

[35] L. Chen, C. Liu, R. Zhou, J. Xu, J. X. Yu, and J. Li, "Finding effective geo-social group for impromptu activities with diverse demands," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 698–708.

[36] M. F. Mokbel, X. Xiong, and W. G. Aref, "Sina: Scalable incremental processing of continuous queries in spatio-temporal databases," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 623–634.

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

[38] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum k-plex problem," *Operations Research*, vol. 59, no. 1, pp. 133–142, 2011.

[39] B. Hou, Z. Wang, Q. Chen, B. Suo, C. Fang, Z. Li, and Z. G. Ives, "Efficient maximal clique enumeration over graph data," *Data Sci. Eng.*, vol. 1, no. 4, pp. 219–230, 2016.

[40] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, 2014, pp. 1311–1322.

[41] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endow.*, vol. 5, no. 9, pp. 812–823, 2012.




Authorized licensed use limited to: Hong Kong Baptist University. Downloaded on February 20,2023 at 03:01:32 UTC from IEEE Xplore. Restrictions apply.