Continuous Geo-Social Group Monitoring in Dynamic LBSNs

Huaijie Zhu, Wei Liu, Jian Yin, Libin Zheng, Xin Huang, Jianliang Xu, *Senior Member, IEEE*, and Wang-Chien Lee

Abstract—*Geo-social group* queries, which return a social cohesive user group with a spatial constraint, have receive significant research interests due to their promising applications for group-based activity planning and scheduling in location-based social networks (LBSNs). However, existing studies on geo-social group queries mostly assume the users are stationary whereas in realistic LBSN application scenarios 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 explore 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. Accordingly, another baseline algorithm, namely *Baseline-CNNE*, is proposed. As these baseline algorithms do not maintain 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. Since IMA requires many times of truss decomposition when processing mutiple-users updates, we propose an improved incremental algorithm, called *improved incremental monitoring algorithm (IIMA)*, which performs truss decompostion only once. Moreover, we design algorithms for handling the social changes that result in insertion/deletion of some edges in the social network. Owing to the challenge in setting, an appropriate monitoring distance, we further study the top N CGSGM problem, which finds top N result groups at each time instance. Finally, we conduct extensive experiments using four real datasets to validate our ideas and evaluate the proposed algorithms.

Index Terms—Location-based services, Geo-social group query, nearest neighbor, continuous queries, monitoring

1 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 [24], [25], [35], [42], [44]. For example, a hotpot restaurant would like to push group coupons to nearby users, who are socially connected with each other, so as to attract them 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 [42].

While much research attention has recently been drawn to geo-social group queries (e.g., [15], [25], [35], [42], [44]), 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 by periodically detecting user groups near the restaurant.

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), in our initial study [43]. 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

- *Huaijie Zhu, Wei Liu, Jian Yin, and Libin Zheng are with Sun Yat-Sen University, Guangzhou, Guangdong 510275, China, and also with the Laboratory of Big Data Analysis and Processing, Guangzhou, Guangdong 510275, China. E-mail: {zhuhuaijie, liuwei, issjyin, zhenglb6}@mail.sysu.edu.cn.*
- *Xin Huang and Jianliang Xu are with Hong Kong Baptist University, Kowloon Tong, Hong Kong. E-mail: {xinhuang, xujl}@comp.hkbu.edu.hk.*
- *Wang-Chien Lee is with Computer Science and Engineering, Penn State University, State College, PA 16802 USA. E-mail: wlee@cse.psu.edu.*

Manuscript received 27 February 2022; revised 28 August 2022; accepted 30 October 2022. Date of publication 2 November 2022; date of current version 21 June 2023.
This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grants 2020B0101100001 and 2018B01010700, in part by the National Natural Science Foundation of China under Grants U1811264,U1811262,U1811261,U1911203,U2001211,U22B2060, 61902438, 61902439 and 62102463, in part by Guangdong Basic and Applied Basic Research Foundation under Grants 2019B1515130001 and 2019A1515011704, and in part by Hong Kong RGC under Grants 12202221, 12200021, C2004-21GF, and 12201018.
(Corresponding Authors: Wei Liu and Jianliang Xu.)
Recommended for acceptance by F. Rusu.
This article has supplementary downloadable material available at https://doi.org/10.1109/TKDE.2022.3218844, provided by the authors.
Digital Object Identifier no. 10.1109/TKDE.2022.3218844

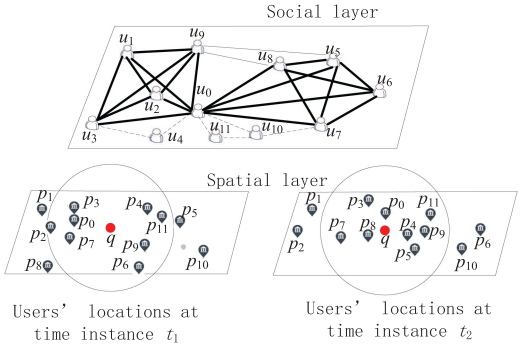


Fig. 1. An example of CGSGM query.

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 Fig. 1, 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 $\{u_0, u_5, u_7, u_9, u_8\}$. Notice that, from 5pm to 9pm, the restaurant may obviously issue many CGSGM queries in between moments as the potential customers keep moving. The scenario and need mandates efficient processing of the query in near-real time. While the aforementioned problem setting is shown to be practically useful, an potential challenge is to set an appropriate monitoring distance d which requires some trials to set right by the restaurant. Thus, we propose an alrernative setting, namely the top N CGSGM problem, which does not require a predefined d, to find top N nearest groups in terms of the group distance to the query location at a time instance.

To support CGSGM, we propose a new query processing framework (as shown in Fig. 2), also called CGSGM for simplicity. It has 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.

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

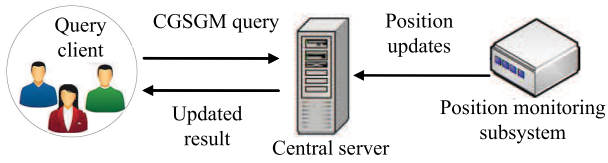


Fig. 2. The CGSGM framework.

is easy to prove that the CGSGM query problem is NP-hard.

To tackle the CGSGM problem, we propose two baseline algorithms, *Baseline-BB* and *Baseline-CNNE* as well as an efficient incremental monitoring algorithm (IMA) in [43]. 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 (i.e., the number of triangles which the edge is involved in), the common neighbors of two nodes, and the neighbors of current users for future user updates and query processing. Since IMA performs truss decomposition many times when processing multiple-users updates, we propose an improved incremental algorithm, called IIMA, which performs truss decompostion only once when handling the multiple-users updates.

To answer the top N CGSGM query, we design an algorithm, namely TOP-N, by incorporating the idea of *distance ordering* to explore the possible combinations of near users, guided by the CNNE strategy explored in Baseline-CNNE to effectively form feasible groups with small distance for prioritized processing. For fast termination of the query processing, we also develop an early terminate condition. *In addition, the social relationship may change for a dynamic user. Thus, we also develop an algorithm for handling the social changes, i.e., removing one edge and inserting one edge.*

In addition to the initial results, this work also extends our initial study in the following aspects, (1) improving the IMA algorithm to answer the CGSGM query (Section 5); (2) formulating and tackling the problem of processing top N CGSGM queries and proposing an algorithm for efficient query processing (Section 6); (3) proposing an algorithm for handling the social changes (Section 7); (4) conducting a more comprehensive performance evaluation which evaluates the the proposed algorithms.

The contributions made in this paper are four-fold:

1. We formalize two new and realistic variant of geo-social queries, *continuous geo-social groups monitoring* (*CGSGM*) query and the top N CGSGM query, over moving users.
2. We propose two baseline algorithms, namely Baseline-BB and Baseline-CNNE, which recompute new geo-social groups from scratch at each time instance for CGSGM.
3. We further develop an improved incremental algorithm for CGSGM, which performs k-truss decomposition only once, by maintaining useful intermediate results for further user updates and query processing.
4. We propose an efficient algorithm, namely TOP-N, for answering the top N CGSGM query. Moreover, we develop an efficient update algorithm to hanlde the social changes in a social network.

5. We conduct extensive experiments to evaluate the proposed algorithms.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 gives some basic definitions and formulates the CGSGM and top N CGSGM problem. Section 4 presents our two baseline algorithms in detail. Section 5 proposes an improved incremental monitoring algorithm. Section 6 extends to process the top N CGSGM query. Section 7 handles the changes of social network. Section 8 reports the experimental results and our findings. Finally, Section 9 concludes the paper with a discussion of future work.

2 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.

2.1 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 [19]. Queries such as "find Alice's nearest petrol stations while she is driving" or "find all the taxi cabs within 1 km from Alice" [32] 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 community. A large number of studies are investigated from various aspects of CSQs, including access methods [13], [31], [33], query algorithms [20], [21], [28] and new query types [5], [16], [22], just to name a few. Over the years, a CSQ has also been called as a moving query [14], [16], [40], a (continuous) spatio-temporal query [3], [31], [36], and a continuous location-based (or location-dependent) query [1], [2], [18], [37], 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. [41] 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 kth NN to q. Mouratidis et al. [30] 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.

2.2 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. [42] 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. [44] 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 [25]. 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. [8] 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. [35] 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. [12] 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. [39] 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. [10] 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 [15] 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. [23] 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. Chen et al. [11] propose a novel geo-social group model, equipped with elegant keyword constraints. Luo et al. [27] study the attribute constrained co-located community (ACOC) search, which returns the smallest community that satisfies three properties: i) structural cohesiveness; ii) spatial co-location; and iii) attribute constraint. While the TCS-SSN [4] is proposed to retrieve a maximal set with high social influence,

small traveling time, covering certain keywords, and containing a query social-network user. In addition, Almaslukh et al. [6] retrieve the most recent k objects that are posted within spatial range R and are posted by u's friends or friends of friends based on a discrete social distance. The k objects are ranked based on the time to retrieve the most recent objects from u's direct friends. Apon et al. [7] investigate the top-k flexible socio-spatial keyword-aware group query, which finds the best k groups of varying sizes around different points of interest (POIs), where the groups are ranked based on the social and textual cohesiveness among members, spatial closeness with the corresponding POI, and the group size. In addition, a social group query problem is proposed to consider the trust among the members of the group [26]. To solve this problem, a multi fuzzy-constrained strong simulation matching model is designed based on multi-constrained simulation.

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. Although Almaslukh et al. [6] study the dynamic change of the query results, it returns the most recent k objects instead of the the social group, which is different from our work. Moreover, the size of geo-social groups returned by CGSGM is fixed, which introduces significant challenges to our work.

3 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 [28]); (2) the query processor periodically pulls the current locations of the objects for query processing in order to update the query answer [29]. In this paper, we adopt the second model and assume that the query processor pulls the current locations of the objects at a scheduled time instance to generate/update the query answer for the query issuer.[1]

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 [34] 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 [9] 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 Fig. 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., $\tau(e) = max_{H \subseteq G}\{\{\tau(H) : e \in E(H)\}$.*

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, and k-edge-connected component community.

3.1 Problem Formulation

In this section, we formally define the CGSGM query.

Problem Statement 1. CGSGM 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

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

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.

Since an appropriate monitoring distance may not easy to be decided by a restaurant, we further study the top N CGSGM problem, which finds top N result groups at each time instance. To rank the result groups, we first define the *group distance* for a group as the measure below.

Definition 7. *(GROUP DISTANCE). Given a k-truss group g of size p, we denote $maxdis(g, q_l)$ as the group distance to query location q_l, and $maxdis(g, q_l) = dis(v, q_l)$ iff $\forall v' \in g$, $dis(v, q_l) \geq dis(v', q_l)$ where $v \neq v'$.*

Problem Statement 2. Top N CGSGM query. 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 $e_{u,v}$. The top N CGSGM query over moving users $\langle N, p, q_l, k\rangle$ continuously monitors top N k-truss groups RG of size p from G between time instance t_1 and time instance t_2 such that each result group $g \in RG, maxdis(g, q_l) \leq maxdis(g', q_l)$ where $g' \notin RG$.

Note that the social changes may occur during the query monitoring period. For simplicity, we first address the problems without social changes and then handle the social changes of the social network (see Section 7).

3.2 K-Truss Index

In this section, we first introduce a k-truss index from [17] 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 [38] 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.

4 Baseline for CGSGM

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 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*) [38] 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.

4.1 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), a basic idea is to enumerate candidate groups of size p by invoking a branch and bound (BB) algorithm.

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.

Through analysis, the branch and bound idea explores many invalid combinations, thus it is inefficient to answer the CGSGM query.

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.*

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 Fig. 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.*

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.*

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 Fig. 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.

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 $\{u_0, u_8, u_9, u_5\}$ 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 8 for the first aspect.

Definition 8. *(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.*

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 the k value 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 $\lfloor\frac{2*n-2}{k}\rfloor$ [10].*

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 $gd(v,u,G) \leq \lfloor\frac{2*n-2}{k}\rfloor$, where $gd(v,u,G)$ denotes the graph distance between u and v in G.*

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 3. 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. In the sequel, CNNE expands the current U_I by selecting one user from remaining users U_R according to the e status. 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.

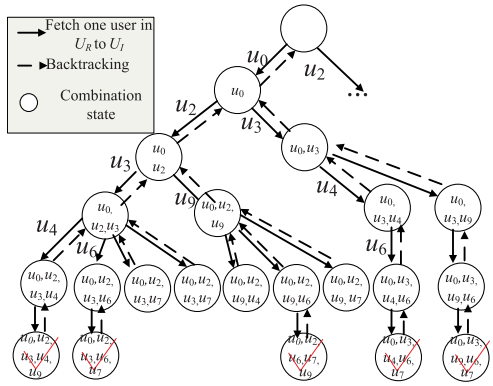


Fig. 3. An example of CNNE.

Fig. 3 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 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.

4.2 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.

5 Improved Incremental Monitoring Algorithm for CGSGM

In this section, we propose an improved incremental monitoring algorithm, namely IIMA, consisting of initial result computation and improved processing of 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.

5.1 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}.

Algorithm 1. 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;

(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 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 2. 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.

5.2 Improved 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.

A direct idea is processing the updates one by one in certain order. Consider the example in Fig. 4 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. Moreover, regarding the IMA algorithm, it spends many time's k-truss decomposition for adding each user. Based on this, we propose an improved multiple user update algorithm, which only does the k-truss decomposition for adding all users.

Algorithm 3. The Improved 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_new;
for each user u ∈ S_new do
  UpdateSupport(sup, u);
  Update the neighbors of w and corresponding common
  neighbors for each w ∈ N_G(u) ∩ S_can with u;
MK ← KTD(S_new, S_can);
for each user u ∈ (S_new ∩ MK) do
  S_kt ← CNNE(u, (MK \ S_new), k);
  insert u into (MK \ S_new);
return S_kt;
```

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, common neighors 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_{new}. After that, we update the supports of involved edges, common neighors and neighbors for those users in S_{new}.

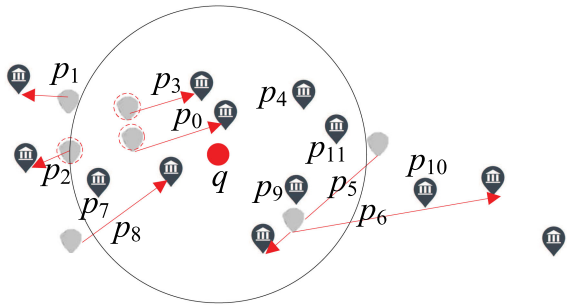


Fig. 4. An example of multiple-users update.

Then obtain the maximal truss MK by performing k-truss decomposition using S_{new} and S_{can}. Notice that we perform k-truss decomposition only once instead of many times of truss decomposition in IMA [43]. Finally, we explore the possible combinations of each user in $(S_{new} \cap MK)$ with $(MK \setminus S_{new})$ by invoking CNNE funtion.

An example of improved processing of multiple user updates for query $\langle d, q_l, 3, 5\rangle$ is shown in Fig. 4. 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 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_{new}. Thus, we need to combine these two users with the users in $MK \setminus S_{new}$. Instead of doing k-truss decomposition with u_5, u_8, respectively (see IMA algorithm in [43]), we do k-truss decompostion with u_5, u_8 together and get the maximal 3-truss $MK = \{u_0, u_3, u_4, u_5, u_8, u_9\}$. We first combine u_5 with $(MK \setminus S_{new}) = \{u_0, u_3, u_4, u_7, u_9\}$ by invoking the CNNE 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.

6 Extension to Top N CGSGM Query

In this section, we propose the approach to monitor the top N geo-social groups, which includes initial result computation and query processing for a new time instance.

6.1 Initial Result Computation

Since the top N geo-social group monitoring returns the top N k-truss groups according to the group distance, obtaining the feasible groups with small distance earlier is a desired way to filter many unqualified groups a lot. For accelerating the exploration of the possible result groups, we design a *distance ordering* to explore the combinations which is also guided by the CNNE strategy to effectively form the feasible groups with small distance earlier. The main idea of distance ordering is to explore the combinations of users by one user at a time according to the distance ordering.

With regard to the distance ordering, we also propose an early termination condition as follows. Let max_dis be the maximal distance of users in the top N k-truss groups

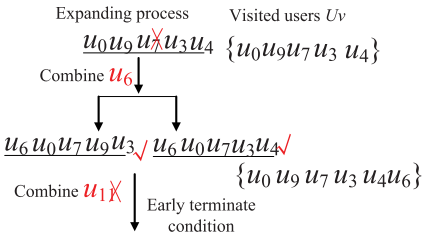


Fig. 5. An example of the IRC_TOP-N algorithm.

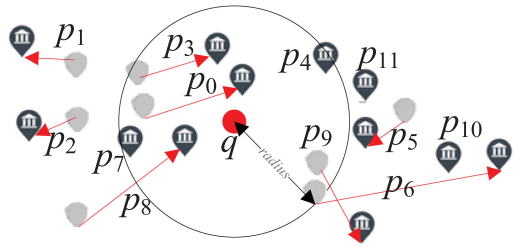


Fig. 6. An example of multiple-users update for top N Geo-social group monitoring.

(denoted by S_{kt}) to query location, i.e., max_dis =Max(dis (u, ql), $u \in S_{kt}$).

Early Terminate Condition. For the current top N result groups achieved so far with the maximal distance max_dis and the current visiting user u, the query process can safely terminate iff dis(u, ql) is greater than max_dis.

The pseudo-code is shown in Algorithm 4. Similar to the initial results computation in Baseline, we filter those unqualified users and sort the remaining users according to their distance to query location in ascending order. After that, we also invoke the k-truss decomposition function and obtain the sorting users of the maximal k-truss stored in a min-heap H_MT (Line 4). Let U_V denote the visited users set. When a new user e is coming, we first check whether its distance to query location is greater than max_dis. If it is, the query processing terminates. Otherwise, we continue to check whether the size of U_V is equal to p-1. If it is, e is inserted into U_V. Then we check whether U_V is a k-truss and update the result if possible (Lines 9-12). If the size of U_V is smaller than p-1, we put e into U_V (Lines 13-14). Otherwise, we invoke function CNNE, to explore the combinations of the new user (i.e., e) with the users in U_V (Lines 15-17).

Algorithm 4. Initial Result Computation for TOP-N (IRC_TOP-N)

Input: A social graph $G = (V, E)$, a query CGSGM $\langle n, q_l, k, p\rangle$, users' location array A_{loc} at the initial time instance,
Output: The top n k-truss groups S_{kt}

```
1  for each user u ∈ A_loc do
2    if τ(u) ≥ k then
3      Insert u into sorted candidate set sortingS_can;
4  H_MT ← KTD(sortingS_can, k);
5  while H_MT ≠ ∅ do
6    de-heap the top entry (e, key) of H_MT;
7    if dist(e, ql) ≥ max_dis then
8      break;
9    if |U_V| == p − 1 then
10     trus insert e into U_V;
11     if U_V is a k-truss then
12       updateRS(S_kt, max_dis, U_V);
13   else if |U_V| < p-1 then
14     insert e into U_V;
15   else
16     CNNE(e, U_V);
17     insert e into U_V;
18 return S_kt;
```

Fig. 5 shows an example for illustrating the IRC_TOP-N algorithm for query $\langle 2, q_l, 3, 5\rangle$. From Fig. 1, we get the sorted maximal 3-truss is $\{u_0, u_9, u_7, u_3, u_4, u_6, u_{11}, u_2, u_1, u_5, u_8, u_{10}\}$. According to the distance ordering, we first fetch the top 5 users U_V=$\{u_0, u_9, u_7, u_3, u_4\}$ and we check whether U_V is a 3-truss. Unfortunately, U_V is not a 3-truss. After that, u_6 is examined. Different from the Baseline, we explore the combinations of u_6 with the current U_V using CNNE function. By selecting a vertex from U_V to combine u_6 according to the CNNE strategy, we first consider the neighbor u_0 of u_6, and the intermediate solution U_I becomes $\{u_0, u_6\}$. Then we use CNNE to decide a common neighbor of e(u_0, u_6), and u_7 is selected. Now U_I becomes $\{u_0, u_6, u_7\}$. At this time, all the edges' support is not less than 1. In the sequel, we select one neighbor u_9 of $\{u_0, u_6, u_7\}$ and U_I becomes $\{u_0, u_6, u_7, u_9\}$. After that, we choose one common neighbor of e(u_0, u_9), i.e., u_3, and we find the 1st top result group $\{u_0, u_6, u_7, u_9, u_3\}$. Similarly, we find the 2nd top result group $\{u_0, u_6, u_7, u_3, u_4\}$ with the maximal distance max_dist = dis(u_6, ql). According to the distance ordering, u_{11} is coming and the distance of u_{11} to ql is greater than max_dist. The query processing terminates.

6.2 Top N Group Monitoring for a New Time Instance

Different from IMA to handle CGSGM query, we maintain the support, common neighbor, neighbor of users in the circle centered at query location with the radius max_dist, which is the maximal distance of users in the top N geo-social groups at the previous time instance.

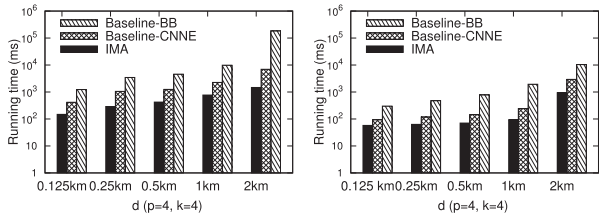
When updating the query results for a new time instance, we first utilize the IIMA algorithm to find the result groups. However, there will be two cases after invoking IIMA algorithm. (1) Top N result groups have been found. For this case, the query processing terminates. (2) We have not found N result groups. With regard to this case, we continue to examine the users outside the circle to explore the possible result groups according to the distance ordering. Take Fig. 6 as an example. Similarly, we utilize the IIMA algorithm to find one result group $\{u_0, u_3, u_8, u_4, u_7\}$ for query $\langle 2, q_l, 3, 5\rangle$. Since there is no enough groups for query, we continue to examine user u_{11} according to this distance ordering to query location. Since there is no vaid common neighbor of e(u_0, u_{11}), we terminate the combination of u_{11}. Then, u_5 is examined. We find top 2 result group $\{u_5, u_0, u_8, u_3, u_4\}$. At this time, we can safely terminate the query processing since the group distance of the remaining combinations will be not less than dis(u_5, ql).

7 Handling the Changes of Social Network

For dealing with a more general case that the social network may change over time, we discuss two main operations, namely removing one edge (not friend any more) and

TABLE 1
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, km)	0.125, **0.25**, 0.5, 1, 2
interval of time instances (minutes)	0.5, **1**, 2, 4, 8



(a) Gowalla dataset (b) Brightkite dataset

Fig. 7. Performance versus query distance d.

inserting one edge (new friend). Since deleting one node can be divide into deleting the neighbor edges of the node and adding one node can be regarded as inserting the neighbor edges and adding one node's information, we here omit to discussing adding/deleting nodes. For these two operations, we first discuss how to update the k-truss index. Then, we study how to update the maintained information.

7.1 Updating the k-Truss Index

For one edge to be removed, we remove it from the hash table and find its all common neighbors. Then we update the support of the common neighbors by reducing one. However, updating the trussness of one edge is complex. First, we should decide which edges' trussness is affected when removing one edge. According to the trussness definition, only the edges whose trussness is less than the trussness of the removed edge (denoted as $\tau(e_{remove})$) are affected because only the edges whose trussness greater than k are kept during doing k-truss decomposition. A feasible way is to do the truss decomposition from k=3 to $\tau(e_{remove})$ to update the edges' trussness.

In contrast, for one edge to be added, we add it into the hash table and also find its common neighbor, update the support of the new edge by assigning the number of common neighbors (denoted as $sup(e_{new})$) and the support of other two edges is added by one. For the trussness of the new edge, it can be at most $sup(e_{new})$+2. Thus, only the edges whose trussness is less than $sup(e_{new})$+2 are affected after adding this new edge. Then we do the truss decomposition from k=3 to $sup(e_{new})$+2 to update the edges' trussness.

7.2 Updating the Maintained Information

For one edge to be removed, there are three cases. i) Both the two nodes of the edge are in the maintained information (e.g., $e(u_7, u_0)$), we remove it from the maintained information and update the neighbors and common neighbors. Then we also find the edge's commnon neighbors in the maintained information and update the corresponding support. ii) Both the two nodes of the edge are not in the maintained information (e.g., $e(u_1, u_2)$). Since deleting one edge may affect the support of other edges and vertex in the maintained information, we check whether their support is not less than k-2 and remove the invalid edges and vertexes. iii) One node v_s of the edge is in the maintained infromation, while the other node v_d of the edge is not (e.g., $e(u_1, u_3)$). While for adding one edge, we first check whether its two nodes are in the maintained information. If it is, we find all the common neighbors in the maintained information and assign the support of this edge as the number of common neighbors. At the same time, we update the support of the common neighbors and the corresponding edges. Otherwise, the nodes of this edge may be invalid. After adding this edge, we check whether they become valid (i.e., support no less than k-2).

8 Experiments

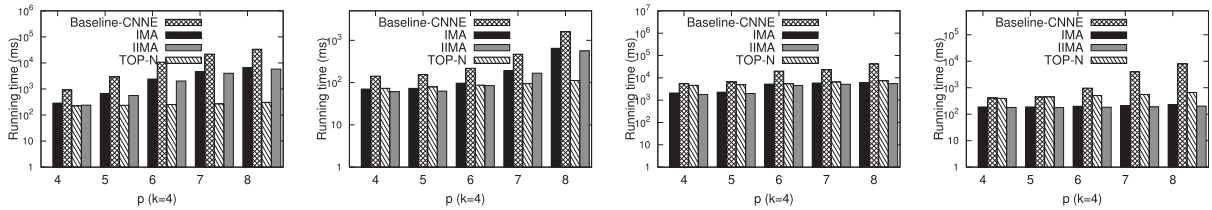
In this section, we evaluate the performance of the proposed algorithms for the CGSGM query. All the algorithms are implemented in C++, while the experiments are conducted on an Intel Core i5 2.3 GHz PC with 16GB RAM.

We conduct experiments on four real datasets. We simulate the user movements based on these datasets.

- **Gowalla**. It consists of 196,585 nodes and 138,337 edges for the social network. The percentage of moving users is 4.89% per minute.[2]
- **Brightkite**. It contains 58,228 users and 214,078 edges. The percentage of moving users is 6.86% per minute.[2]
- **Foursquare**. It consists of 2,146,576 nodes and 8,919,127 edges for the social network. The percentage of moving users is 5.46% per minute.[2]
- **Flickr**. It consists of 214,698 nodes and 2,096,306 edges for the social network. The percentage of moving users is 6.98% per minute.[3]

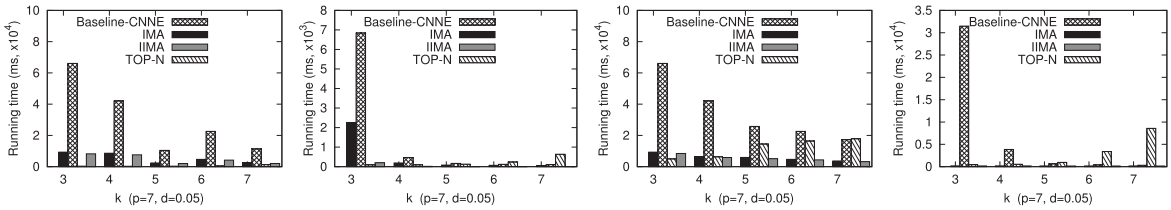
We conduct a performance evaluation on the efficiency of the CGSGM algorithms – we compare the latency of the proposed CGSGM algorithms under various parameters (summarized in Table 1, numbers in bold are the default settings). The parameters are set according to a typical restaurant marketing application, where the group size is no more than 8, the social constraint is no more than 7, and the distance constraint is no farther than 2 km. 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; and (d) interval of time instances. 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.

2. https://snap.stanford.edu/data/index.html.
3. https://www.comp.hkbu.edu.hk/db/book/community_search.html.



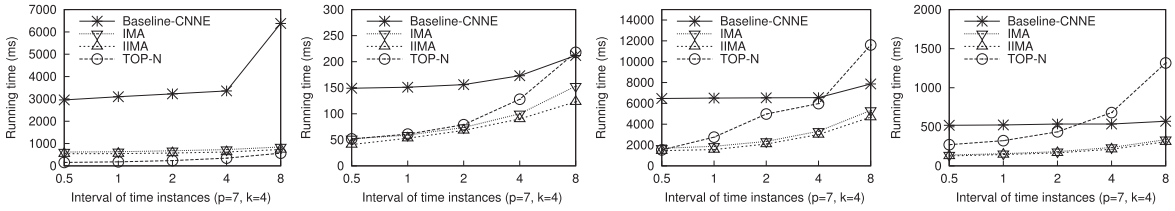
(a) Gowalla dataset (b) Brightkite dataset (c) Foursquare dataset (d) Flicker dataset

Fig. 8. Performance versus query group size p.



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

Fig. 9. Performance versus social constraint k.



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

Fig. 10. Performance versus interval of time instances.

8.1 Efficiency of CGSGM Algorithms

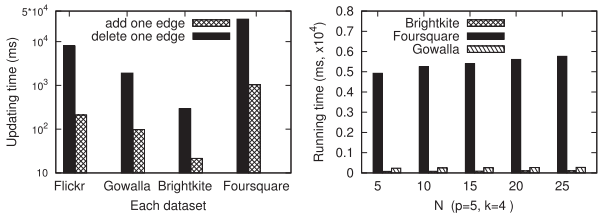
In this section, we compare the efficiency of the proposed IMA and its imrpoved version (IIMA) with two baseline algorithms, Baseline-BB and Baseline-CNNE, as well as the top N algorithm (TOP-N) by setting N=5 in default. Accordingly, we test the four CGSGM algorithms using different datasets as described above. We take an offline approach to obtain the common neighbors for each edge in our implementation.

Effect of Distance Threshold. We first test the performance of the four algorithms by varying query distance d. Fig. 7 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, IIMA, Baseline-CNNE and TOP-N for processing CGSGM queries by increasing the group size p. Fig. 8a 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. Moreover, IIMA outperforms IMA with 10%, improvement. 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 IIMA and IMA scale better than Baseline-CNNE by increasing the group size. The results on other three datasets are similar to that on the Gowalla dataset.

Effect of Social Constraint k. Fig. 9 compares the performance of three CGSGM algorithms and the TOP-N algorithm 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. IIMA is faster than IMA becuase IMA requires to perform many time's k-truss decompositon for all the adding users while IIMA does the k-truss decompositon only once for all adding users, which saves a lot of time. Moreover, the processing time of both four 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 Fig. 9b, Flicker dataset in Fig. 9d and Foursquare dataset in Fig. 9c show the similar trend with the result on the Gowalla dataset.

Effect of Interval of Time Instances. In addition, we compare the four CGSGM algorithms by varying the interval of time



(a) Handling social changes (b) Varying N value

Fig. 11. Performance of handling social changes and varying N values.

instances and show the results in Fig. 10. 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 Fig. 10a show the superiority of IIMA and IMA over the Baseline-CNN algorithm. The search time of both IMA and IIMA 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 interval of time instances becomes longer, which means more user updates, the query time of IIMA, IMA and Baseline-CNN becomes longer. Similar results on Brightkite and Foursquare dataset are observed in Figs. 10b and 10d.

Effect of Handling Social Changes and Varying N Values. At last, we also do the experiments to test the updating time of handling the social changes including inserting one edge and removing one edge on four datatsets. The social network changes apply to both of the two problems. At each time instance, we first handle the social changes and then processing the query. During the query processing, we assume the social network is static. Thus, handling social changes only updates the k-truss index and maintained information. The corresponding results are shown in Fig. 11a. As shown, the updating time for removing one edge takes longer than the time for inserting one edge on each dataset because removing one edge involves in more edges to be updated. Fig. 11b shows the results by varying N value to test the Top-N algorithm on three datasets. As expected, when the N value becomes larger, the algorithm requires more time to return the top N result groups.

9 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 improved incremental algrithm, namely IIMA, 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. We also formalize and investigate the top N CGSGM query. Moreover, for handling the social changes in a social network, we develop the corresponding algorithm. At last, we conduct a comprehensive performance evaluation using three real datasets to validate our ideas and compare the proposed algorithms.

References

[1] 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.
[2] I. Afyouni, C. Ray, S. Ilarri, and C. Claramunt, "Algorithms for continuous location-dependent and context-aware queries in indoor environments," in *Proc. 20th Int. Conf. Adv. Geographic Inf. Syst.*, 2012, pp. 329–338.
[3] P. K. Agarwal, L. Arge, and J. Erickson, "Indexing moving points," *J. Comput. Syst. Sci.*, vol. 66, no. 1, pp. 207–243, 2003.
[4] A. Al-Baghdadi and X. Lian, "Topic-based community search over spatial-social networks," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2104–2117, 2020.
[5] 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.
[6] A. Almaslukh, Y. Kang, and A. Magdy, "Temporal geo-social personalized keyword search over streaming data," *ACM Trans. Spatial Algorithms Syst.*, vol. 7, no. 4, pp. 20:1–20:28, 2021.
[7] S. H. Apon, M. E. Ali, B. Ghosh, and T. Sellis, "Social-spatial group queries with keywords," *ACM Trans. Spatial Algorithms Syst.*, vol. 8, no. 1, pp. 1–32, 2022.
[8] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *Proc. VLDB Endowment*, vol. 6, no. 10, pp. 913–924, 2013.
[9] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum K-plex problem," *Operations Res.*, vol. 59, no. 1, pp. 133–142, 2011.
[10] 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.
[11] 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 *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 698–708.
[12] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *Proc. VLDB Endowment*, vol. 10, pp. 709–720, 2017.
[13] B. Gedik and L. Liu, "MobiEyes: Distributed processing of continuously moving queries on moving objects in a mobile system," in *Proc. Int. Conf. Extending Database Technol.*, 2004, pp. 67–87.
[14] B. Gedik and L. Liu, "Mobieyes: A distributed location monitoring service using moving location queries," *IEEE Trans. Mobile Comput.*, vol. 5, no. 10, pp. 1384–1402, Oct. 2006.
[15] B. Ghosh, M. E. Ali, F. M. Choudhury, S. H. Apon, T. Sellis, and J. Li, "The flexible socio spatial group queries," *Proc. VLDB Endowment*, vol. 12, pp. 99–111, 2018.
[16] W. Huang, G. Li, K. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 932–941.
[17] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1311–1322.
[18] S. Ilarri, E. Mena, and A. Illarramendi, "Location-dependent queries in mobile contexts: Distributed processing using mobile agents," *IEEE Trans. Mobile Comput.*, vol. 5, no. 8, pp. 1029–1043, Aug. 2006.
[19] T. Imielinski and B. R. Badrinath, "Querying in highly mobile distributed environments," in *Proc. 18th Int. Conf. Very Large Data Bases*, 1992, pp. 41–52.
[20] G. S. Iwerks, H. Samet, and K. P. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *Proc. 29th Int. Conf. Very Large Data Bases*, 2003, pp. 512–523.
[21] D. V. Kalashnikov, S. Prabhakar, S. E. Hambrusch, and W. G. Aref, "Efficient evaluation of continuous range queries on moving objects," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2002, pp. 731–740.
[22] 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, Feb. 2009.
[23] Q. Li, Y. Zhu, and J. X. Yu, "Skyline cohesive group queries in large road-social networks," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 397–408.
[24] Y. Li, R. Chen, L. Chen, and J. Xu, "Towards social-aware ridesharing group query services," *IEEE Trans. Serv. Comput.*, vol. 10, no. 4, pp. 646–659, Jul./Aug. 2017.

[25] Y. Li, R. Chen, J. Xu, Q. Huang, H. Hu, and B. Choi, "Geo-social K-cover group queries for collaborative spatial computing," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 10, pp. 2729–2742, Oct. 2015.
[26] G. Liu, L. Li, G. Liu, and X. Wu, "Social group query based on multi-fuzzy-constrained strong simulation," *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 3, pp. 54:1–54:27, 2022.
[27] J. Luo, X. Cao, X. Xie, Q. Qu, Z. Xu, and C. S. Jensen, "Efficient attribute-constrained co-located community search," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1201–1212.
[28] 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.
[29] M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: Scalable incremental processing of continuous queries in spatio-temporal databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 623–634.
[30] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 634–645.
[31] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of moving object trajectories," in *Proc. 26th Int. Conf. Very Large Data Bases*, 2000, pp. 395–406.
[32] 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. Surveys*, vol. 51, no. 3, pp. 64:1–64:39, 2018.
[33] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. López, "Indexing the positions of continuously moving objects," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 331–342.
[34] S. B. Seidman, "Network structure and minimum degree," *Social Netw.*, vol. 5, no. 3, pp. 269–287, 1983.
[35] C. Shen, D. Yang, L. Huang, W. Lee, and M. Chen, "Socio-spatial group queries for impromptu activity planning," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 196–210, Jan. 2016.
[36] Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2002, pp. 334–345.
[37] 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, Jul. 2011.
[38] J. Wang and J. Cheng, "Truss decomposition in massive networks," *VLDB*, vol. 5, no. 9, pp. 812–823, 2012.
[39] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin, "Efficient computing of radius-bounded K-cores," *Proc. IEEE 36th Int. Conf. Data Eng.*, 2018, pp. 233–244.
[40] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-K spatial keyword query processing," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2011, pp. 541–552.
[41] X. Xiong, M. F. Mokbel, and W. G. Aref, "SEA-CNN: Scalable processing of continuous K-nearest neighbor queries in spatio-temporal databases," in *Proc. IEEE 21st Int. Conf. Data Eng.*, 2005, pp. 643–654.
[42] D. Yang, C. Shen, W. Lee, and M. Chen, "On socio-spatial group query for location-based social networks," *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2012, pp. 949–957.
[43] H. Zhu et al., "Continuous geo-social group monitoring over moving users," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2022, pp. 312–324.
[44] Q. Zhu, H. Hu, C. Xu, J. Xu, and W. Lee, "Geo-social group queries with minimum acquaintance constraint," *VLDB J.*, vol. 26, no. 5, pp. 709–727, 2017.

Huaijie Zhu received the BSc degree from the Information Science Department, Kunming University of Science and Technology, the MSc degree from the Computer Science Department, Northeastern University, and the PhD degree in computer science from Northeastern University, China, in 2018. He is currently an associate professor with Sun Yat-Sen university. His research interests include spatial database, and data privacy.

Wei Liu received the BS degree in computer science from Shanghai University, in 2010, the MS degree in computer science from the South China University of Technology, in 2013, and the PhD degree in computer science from Sun Yat-Sen University, China, in 2018. He is doing a postdoctoral fellow with Sun Yat-sen University from 2018. His current research interests include the areas of recommendation system, and user behavior learning in location-based social network.

Jian Yin received the BS, MS, and PhD degrees in computer science from Wuhan University, China, in 1989, 1991, and 1994, respectively. He is a professor with Data and Computer Science School. He has published more than 100 refereed journal and conference papers. His current research interests include the areas of data mining, artificial intelligence, and machine learning. He is a senior member of China Computer Federation.

Libin Zheng received the BS degree in computer science and engineering from the South China University of Technology, China, in 2015. He is now associate professor with Sun Yat-sen University. His research interests include spatial crowdsourcing, shared-mobility, and classification crowdsourcing.

Xin Huang received the PhD degree from the Chinese University of Hong Kong (CUHK), in 2014. He is currently an assistant professor with Hong Kong Baptist University. His research interests include mainly focus on graph data management and mining.

Jianliang Xu (Senior Member, IEEE) received the BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, in 1998, and the PhD degree in computer science from the Hong the Kong University of Science and Technology, in 2002. He is a professor with the Department of Computer Science, Hong Kong Baptist University. He is an associate editor of *IEEE Transactions on Knowledge and Data Engineering* (TKDE).

Wang-Chien Lee received the BSc degree from the Information Science Department, National Chiao Tung University Taiwan, the MSc degree from the Computer Science Department, Indiana University, Bloomington, and the PhD degree from the Computer and Information Science Department, Ohio State University. He is an associate professor of computer science and engineering with Pennsylvania State University, leading the Pervasive Data Access research group.