Temporal Social Network: Group Query Processing*

Xiaoying Chen[1] Chong Zhang[2] Yanli Hu[3] Bin Ge[4] Weidong Xiao[5]
Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology, Changsha 410073, P.R.China
and
Collaborative Innovation Center of Geospatial Technology, China
{[1]chenxiaoying1991, [2]leocheung8286}@yahoo.com [3]smilelife1979@163.com
[4]gebin1978@gmail.com [5]wilsonshaw@vip.sina.com

***Abstract*—With the development of on-line social networks, more and more users and sellers are interested in group analytics which provides insights into common interests with various relationships. This paper addresses this problem from temporal aspect, i.e., temporal group query (TGQ) which gives people the historical view for group forming and changing. To efficiently achieve our goal, we propose two structures to index temporal social network (TSN), and then a simple naive searching method is designed to process the TGQ, after that we argue a more efficient approach can be implemented by update operations instead of iterative graph generations, which we call optimized method. We conduct experiments on real-synthetic dataset, and the results show that our indexes and searching algorithms are capable, and optimized method is much more efficient than the naive one.**

I. Introduction

Due to the increasing popularity of social networks and vast amount of information in them, there have been many efforts in enhancing web search based on social data. Social networks contain data about the network, i.e., data about users and their social relations, data about the activities that users participate in. Besides the graph structure, another important dimension of social network is dynamic attribute. For instance, logins or logouts of a user represents changes in the users' on-line status. Also, new content is added through user activities (such as posting text, photo) representing respective changes in the users' interests. This temporal aspect of information in social network would influence social network query either explicitly by enabling users to query for particular time points or periods, or implicitly by providing the fresher results, e.g., find some users who are active during a certain period, or find some activities posted at some time. Such queries add temporal axis to user requirement, we call it temporal social network query.

In our previous work, we introduced temporal social network, and address three query types, namely Friends of Interesting Activities (FIA) query, Users of Time Filter (UTF) query and Group of Users with Relationship Duration (GURD) query. In GURD query, it aims to find a set of groups, where the number of users in each group is equal to a given number, and average intimate degree – which is measured by average relationship duration – of it satisfies a given value, and all the members of it have taken part in some given activities. In this paper, we argue that GURD is not sufficient for analyzing the changes of groups, which is an essential module for temporal social networks analytics. Thus, we intend to extend GURD query to a more generalized and applicable one to adapt more temporal queries in real world. Here, we call it Temporal Group Query (TGQ).

TGQ is essential, for instance, when the system analyst would like to observe and analyze the formation and changes of groups, historically. For example, they aim to find a set of groups, each group with all member participating in the activity labeled with '*presidential election*' during last two weeks, and average on-line duration is not less than 24 hours during last two weeks. The result of this query should be appropriately in the form: $\{< g_1=\{u_1, u_2, \dots\}, t_1, t_2>, < \dots >, \dots\}$, where g_1 is a satisfying group valid during $[t_1, t_2]$.

For efficiently answer TGQ query, we design two indexes, one is called Temporal Activity tree (TA-tree) indexing participating time and keywords' activity , the other is Temporal Friendship tree (TF-tree) indexing versions of relationships. We first devise an approach naively iterate each relationship changing time point, and construct the satisfying group. However, we argue that the naive method is not efficient, thus we design a more efficient algorithm to accelerate the processing, in particular, it is a series update operations to an initial graphs and check which updated graph is satisfied. Due to less time for constructing connected graph, it is more efficient than the naive one. In this paper, we make the following contributions:

- We propose a more applicable query type, temporal group query, to answer group query in temporal social network.
- We design two approaches to process TGQ, one is naive iterative, the other is an optimized one.
- We conduct experiments on real-synthetic hybrid dataset, and results show our method is efficient.

The rest of this paper is organized as follows. Related works are surveyed in section 2. Problem is formally defined in section 3. In section 4, two index TA-tree and TF-tree are presented, followed by query processing in section 5. In section 6, we carry out our experiments. Finally, section 7 concludes the paper with directions for future works.

*This work is supported by NSF of China grant 61303062 and 61302144.

II. Related Works

Some of the relevant works of temporal social network study focus on efficient algorithms and data structure for searching data. And some others propose some typical kinds of temporal-social query that can be applied to life application. We have proposed three kinds of queries, namely FIA, UTF and GURD query in [1]. GURD query is a kind of temporal group query, which aims to find a set of groups, where the number of users in each group is equal to a given number, and average intimate degree of it satisfies a given value, and all the members of it have taken part in some given activities. We propose two index structures, TUR-tree and TUA-tree for accelerating query process. Algorithms of query processing for the three queries are finally proposed.

There are many other works introduce many kinds of group queries, such as [2], [3], [4]. Social-Temporal Group Query (STGQ) [2], which considers the available time of candidate attendees and their social relation, aims to find activity time and attendees with minimum total distance to the initiator. However, in STSG, the time dimension is only considered as available time for attendees, which can be dealt with after Social-Group Query (SGQ).

In the other hand, adding the spatial dimension into social network would bring about different group queries. Socio-Spatial Group Query (SSGQ) [3] finds a group of attendees close to rally point and ensure that the selected attendees have a good social relationship to create a good atmosphere in the activity. There are many similar queries, for example, Circle of Friends (CoF) [4] finds a group of friends whose members are close to each other both socially and geographically. These two query problem are both NP-hard (because social restrictions diverse), but without time dimension.

As there are many kinds of temporal social network queries, it is impossible to study all of them. So Kostas Stefanidis et.al.[5] define a logical algebra that provides a set of operators required for temporal social network query evaluation. However, this work only provides a logical algebra without any storage model and query processing detail which is the important factor to the effectiveness of this framework. As we have introduced above, there are few previous studies consider the efficient query processing in temporal social group query in historical view.

III. Problem Definitions

Given an undirected graph G=(V, E), V=U∪A. Each vertex u_i in U represents a user in the community and is associated with a time interval [ut_s, ut_e), meaning the valid period during which u_i exists (or being logon status) in community, and [ut_s, $*$) means u_i is still in the community now. For a given social network graph G, there is also a set A⊂V, in this paper, we use term *activity* to denote social event in the social network, e.g., publishing a post, sharing a link and etc. Without loss generality, each activity can be represented as $< aid, W_a >$, where aid is the activity identifier, W_a is a keyword set to describe the activity.

Each edge (v_i, v_j) in E has two forms: (u_i, u_j) and (u_i, a_j). (u_i, u_j) represents a friend relationship between user u_i and u_j, and it is also associated with a time interval [et_s, et_e), in which et_s means the time when the relationship is established and et_e means the time when the relationship is removed.

Each u_i in G can connect to an activity a_k in A, which means that u_i gets involved in a_k, formed as (u_i, a_k). We use term *participation* to describe relationship between user and activity, e.g., in social network, user may post some texts, or forward other's post, or share a link of web page, or comment other's activity, or add some activity into favorite, in general, we call these actions as participations.

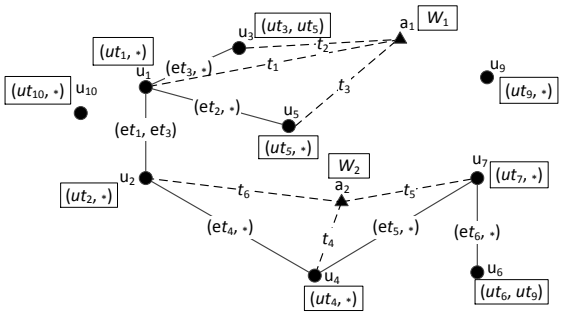


Fig. 1. Temporal Social Network Example

Figure 1 illustrates a temporal social network, where users and activities are plotted as dots and triangles, respectively. The relationship are divided into two categories: user-to-user (solid line) and user-to-activity (dashed line). And the label on edge indicates time interval of relationship or time stamp when a user participate in an activity. For instance, user u_1 logins community at time ut_1 and does not logout at current, and user u_1 and u_2 become friends at et_1 and unfriend each other at et_3, and u_4, u_7 and u_2 participate in a_2 at t_4, t_5 and t_6, respectively.

Then we formally give definition of TGQ, a TGQ (W, [t_s, t_e], t_{ol}) aims to find a set of groups, in which, each group is a connected graph, and all members have participated in the activities with keywords in W during period [t_s, t_e], and the average on-line duration (AOD) (during [t_s, t_e]) of the members is not less t_{ol}. Here on-line duration (AOD) is defined as:

$$AOD = \frac{\sum_{i=1}^{n} onlineduration(u_i)}{n} \quad (1)$$

where n is the cardinality of the group, $onlineduration(u_i)$ calculates on-line duration of user u_i.

IV. Indexes Design and Functions

We believe that for a TGQ (W, [t_s, t_e], t_{ol}), two kinds of temporal predicates should be taken into consideration of building indexes. One is the temporal predicate [t_s, t_e] for participation time, and the other is friendship valid period predicate for generating groups with valid period. Thus, we design two indexes to accelerate query processing. One index is for finding which users participating in the activities with keywords in W during [t_s, t_e], and the other one aims to

process the query – given a time period, find pairs of users who are friends during the period.

The first index we call it Temporal Activity tree (TA-tree), which is actually a B^+-tree injected with Bloom Filter[6]. In particular, the data item (entry in leaf node) to be indexed is in the form $<u_i, t_p, a_k, W_{a_k}>$, where u_i is a user identifier, t_p represents the time user u_i participates in activity a_k, and W_{a_k} is the keyword set describing a_k. B^+-tree is built according to the key t_p, and the keyword sets of all entries in each leaf node constitute a Bloom Filter BF, and BF serves as filter with pointers in internal nodes. Figure 2 illustrates an example of TA-tree. We use a querying function to represent retrieval function of TA-tree, R=$TARetrv([t_s, t_e], W)$, which means to find user set R participating in the activities with keywords in W during $[t_s, t_e]$.

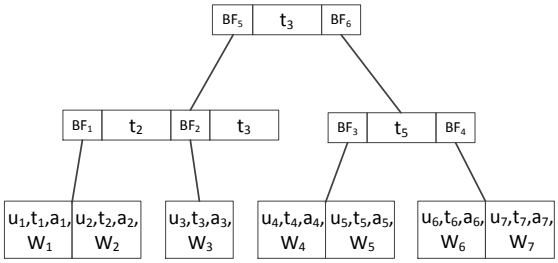


Fig. 2. TA-Tree Example

The second index is Temporal Friendship tree (TF-tree), which is a kind of MVB-tree[7], indexing the temporal friend relationship. In particular, data item to be indexed is in the form $<u_i|u_j, [t_f, t_u]>$, where $u_i|u_j$ is concatenation of a pair of friends serving as the search key, and $[t_f, t_u]$ is the valid friendship period of u_i and u_j. Thus, TF-tree's function is to find which pairs of users are valid friends during the given time period, and the retrieval function can be represented as R=$TFRetrv([t_f, t_u])$, which means to retrieve pairs of friends associated with corresponding valid time period during $[t_f, t_u]$.

V. Query Processing

In this section, we handle the query processing for TGQ. First we propose a straightforward approach, after that, we optimize it to improve the efficiency.

A. Naive Searching

For a TGQ (W, $[t_s, t_e]$, t_{ol}), the basic workflow is as following: firstly, TA-tree is used to retrieve the user set (say U_c) participating in the activities containing the keyword set W during $[t_s, t_e]$, then for each candidate user u_i in U_c, sum of on-line duration during $[t_s, t_e]$ is calculated. Then, the difficulty lies in how to return the satisfied group with valid time period. Through observation, we get that the group's valid period depends on each relationship's changing time, e.g., a temporal interval for a relationship is $[t_f, t_u]$, which would contribute the valid period of the group. Thus we can use TF-tree to retrieve all pairs of friends during $[t_s, t_e]$, say F_c, and we can generate pairs of friends in U_c associated with valid time period by intersecting U_c and F_c, after that, for each time point when friendship change happens in U_c, we calculate the satisfied snapshots of groups (i.e., connected graph), and then form the results.

For accelerating the processing, we use a max-queue storing users in U_c sorted by on-line duration, thus we can terminate the loop as early as possible. In particular, on each time point t_i, we calculate a set of snapshot connected graph whose AOD is larger than or equal to t_{ol}, i.e., top element u_{top} of the max-queue is popped and checked whether on-line duration is not less than t_{ol}, if so, that is to say it is possible for finding a satisfied connected graph containing u_{top}, then a connected graph G_{t_i} on time t is generated, and check whether AOD of G_{t_i} is larger than or equal to t_{ol}, if so, G_{t_i} is a result, after that, all users in G_{t_i} are removed from the queue, and next top element is popped and the similar processing is continued. Otherwise, i.e., on-line duration of u_{top} is less than t_{ol}, which means it is impossible to find a satisfied connected graph for the elements following u_{top}, so the loop for the queue is terminated and processing is moved to the next time point. Algorithm 1 presents the pseudo-code of naive searching.

Algorithm 1 Naive Searching

Input: q=(W, $[t_s, t_e]$, t_{ol})
Output: $Rlist$
1: U_c=$TARetrv([t_s, t_e], W)$
2: Q=$generateQueue(U_c)$
3: F_c=$TFRetrv([t_s, t_e])$
4: TP=$extractTimePoints(F_c, U_c)$
5: **for** each $tp \in TP$ **do**
6: $replQ$=Q
7: **while** $replQ \neq \phi$ **do**
8: u=$dequeue(replQ)$
9: **if** $u.od \leq t_{ol}$ **then**
10: CQ=$geneCQ(u, tp)$
11: **if** $AOD(CQ) \leq t_{ol}$ **then**
12: $Rlist \leftarrow (CQ, tp)$
13: **end if**
14: **else**
15: break
16: **end if**
17: **end while**
18: **end for**
19: **return** $Rlist$

For example, after retrieving the users that satisfied the keywords and time constristant, U_c=$\{u_1,u_2,...,u_{15}\}$ is found, and t_{ol}=300. The relationships between these users, F_c is showed in Figure 3 with time interval tags. Meanwhile, a timeline containing t_i, and the sum of on-line duration of each u_i is also showed in this figure.

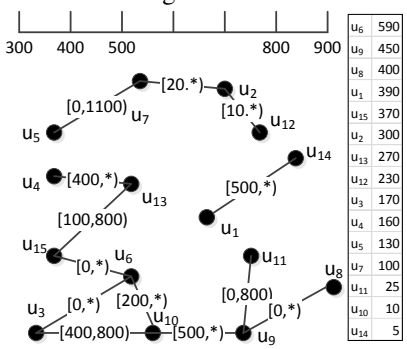


Fig. 3. Example of TGQ

As shown in Figure 4, on time t_1=300, $u_{top} = u_6$ (show in bold), we form a connected graph for u_6, and then check that AOD=382>300, so we add it to the result list. And then $\{u_6, u_3, u_{10}, u_{15}, u_{13}\}$ are removed. u_{top} changes to u_9, we simply form a a connected graph for u_9, and check that AOD=291.7<300, this result does not satisfy. Until the $u_{top} = u_2$, the on-line duration of u_2=300, is not less than t_{ol}. So the processing is jumped to the next time point t_2=400. Due to space limitations we do not show the processing at time 500 and 800.

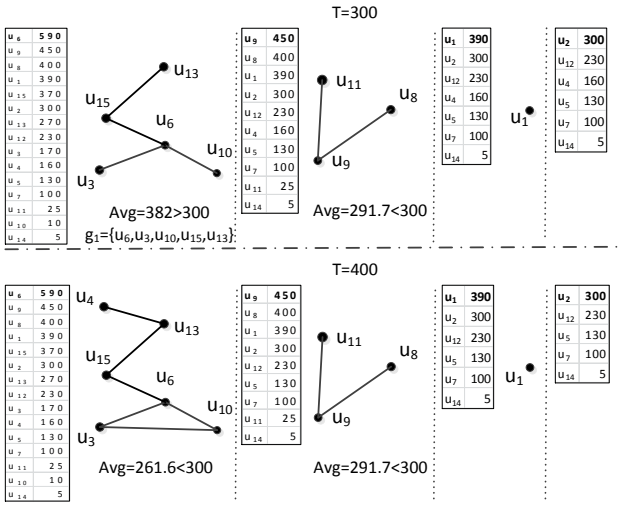


Fig. 4. Naive Searching Example

B. Optimized Processing

We optimize the naive searching in this section, through observation, we can see that, it is not necessary to construct connected graphs at each time point. On the contrary, all the connected graphs can be constructed initially, and at each time point, according to the changes of relationship, we update the corresponding graph and make judgment whether it is a satisfied group. In particular, we argue that the key techniques lie in the update operations to connected graphs. On each time point, there are one or more operations to previous connected graphs, and we distinguish the following cases:

(1) Addition of relationship between nodes both in a connected graph cg. For this case, no judgment need to make, i.e., the results are the same with the previous step.

(2) Addition of relationship between a node in a connected graph and an isolating node $inode$. For such a case, a new connected graph containing $inode$ is generated, which should be inspected whether it is a satisfied group, i.e., whether average on-line duration is not less the given value.

(3) Addition of relationship between two isolating nodes. For such a case, a new connected graph is formed, the processing is the same with case (2).

(4) Addition of relationship between one node in a connected graph and the other node in another connected graph. For such a case, a new connected graph is formed, the processing is the same with case (2).

(5) Removal of relationship between two nodes, resulting no split in a connected graph. For this case, no judgment need to make, and results are unchanged.

(6) Removal of relationship between two nodes, resulting a connected graph split. For this case, the two emerging connected graphs should both be inspected whether they are still satisfied groups.

Unlike the naive searching, the optimized processing only consider the changes of relationship in the graph, also take Figure 3 as an example, the processing is shown in Figure 5. Firstly, we form connected graphs at t_1=300. Then at t_2=400, the changes of relationship are addition of relationship $\{u_4,u_{13}\}$ (case 2) and $\{u_3,u_{10}\}$ (case 1). Adding the relationship $\{u_3,u_{10}\}$ in case 1 have no effect on the previous step, while adding the relationship $\{u_4,u_{13}\}$, a new connected graph is formed. In this case, a new member u_4 (on-line duration is 160) is added to G_{t1} in time 300 which results to AOD changing and G_{t1} is no longer the satisfied result. At t_3=500, the changes of relationship are establishment relationship $\{u_1,u_{14}\}$ (case 3) and $\{u_9,u_{10}\}$ (case 4). At t_4=800, the changes of relationship are removal of relationship $\{u_3,u_{10}\}$ (case 5) and $\{u_{13},u_{15}\}$, $\{u_9,u_{11}\}$ (case 6).

VI. Experimental Evaluation

We implement the indexes and processing algorithms, and experimentally evaluate them on a synthetic-real-hybrid dataset. Due to the fact that there is no real dataset containing temporal information on on-line status, relationship and activity participation. Thus we have to synthetically generate dataset based on some real datasets which contain partial temporal attributes. Table I lists some properties of these datasets. Datasets $Users$ is generated according to the case of YouTube, which owing 3,223,589 users. Then according to birthday information of Users, we randomly produced the login time list of each user, which forms the $login$ dataset. Datasets $Relation$ is produced based on the real dataset[1], which contains 9,375,374 pairs of relation of YouTube. For the dataset of $Activity$, firstly, we download the dataset about food theme on GCZX server, formed in xml file of 25 classes of Amazon commodities, 1759 review on the hotel and some web crawling data, etc. Secondly, we extract the useful text as part of the text of activity, and randomly generated users who involved in this activity and the time participate in the activities.

TABLE I
Dataset Description

Dataset	Record number	Data size
User	3,223,589	287M
Relation	9,375,374	5.53G
Login	3,223,589	2.45G
Activity	6,980,465	2.28G

We implement our algorithms in Java. To make comparison, the naive searching approach is used as a baseline. We vary the query parameters and at each testing point, 10 queries are issued to collect the average results. The experiments are conducted on a DELL server with Intel(R) Xeon(R) 2.40GHz

[1]http://konect.uni-koblenz.de/downloads/tsv/youtube-u-growth.tar.bz2

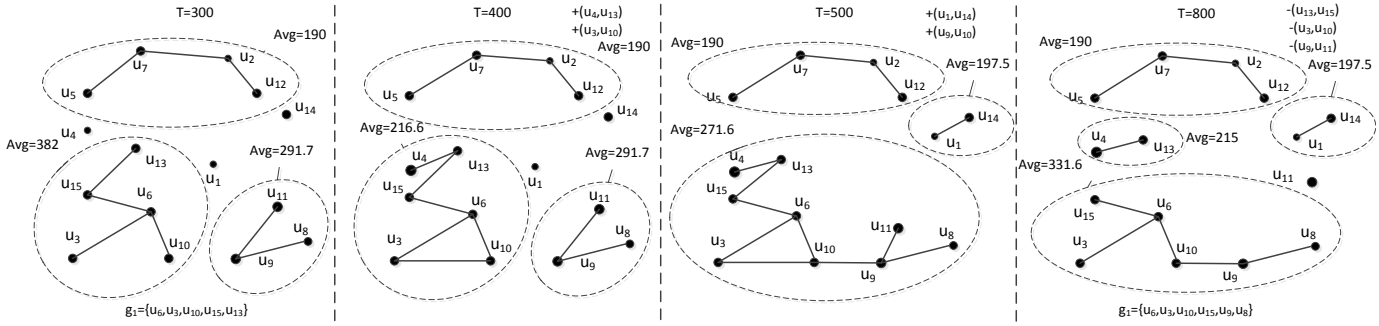


Fig. 5. Optimized Processing Example

processor, 8GB memory and 500GB disk. Table II describes the query parameters.

TABLE II
PARAMETERS IN EXPERIMENT

queries	parameters	domain	defualt
TGQ	t_{ol}	1 - 5	3
	W_q's cardinality	1 - 5	3
	$length_{[t_s,t_e]}$/temporal extent	0.1%-3%	1%

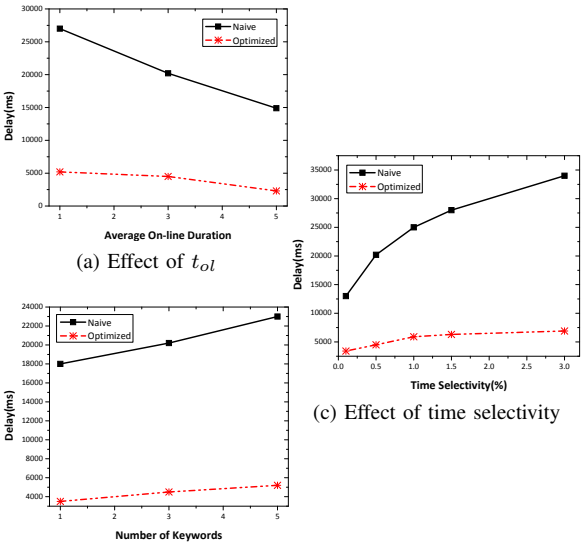
Firstly, we vary t_{ol} to compare the performances of two algorithms. We increase t_{ol} from 1 to 5, while the response time of query decreases (see Figure 6(a)), this is due to our terminating condition, i.e., a larger t_{ol} will terminate loop earlier, thus the processing delay is reduced. For the detail, we can see the effectiveness of our optimization, i.e., the optimized approach utilizes the updates to the graphs, which reduces the simple repeated group forming procedure, thus it cost less time to retrieve the results.

Next, we increase the number of querying keywords to test performances. We can see from Figure 6(b), the response time also increases with the number of keywords. This can be explained that a larger number of keywords would involve more tree nodes in the TA-tree to be traversed, thus more time would be cost. Similarly, the optimized approach outperforms the naive searching.

Figure 6(c) illustrates the results of varying parameter time selectivity, a larger value causes more candidates to be inspected, thus the response time increases correspondingly. And still, the results show that a series of update operations is more efficient than exhausted method.

VII. CONCLUSION

With applications continuously developing, more and more temporal social network group queries will be paid attention. In this paper, we focus on temporal analytics on social group query, and argue Temporal Group Query (TGQ) is useful and applicable. To efficiently address the query, we design two indexing structures to accelerate the query processing, and two processing algorithms, one is simple, the other is optimized, to accomplish the processing. We conduct experiments on real-synthetic hybrid dataset, and the results show our methods are capable and optimized method is efficient. In the future, we would like to study on social group query with geographic attributes.



(a) Effect of t_{ol}

(c) Effect of time selectivity

(b) Effect of keywords

Fig. 6. Experimental Results

ACKNOWLEDGMENT

This work is supported by NSF of China grant 61303062 and 61302144. We would like to thank Prof. Dai and Dr. Hu for helping with the proof.

REFERENCES

[1] X. Chen, C. Zhang, B. Ge, and W. Xiao, "Temporal social network: Storage, indexing and query processing," EDBT, 2016.

[2] D. N. Yang, Y. L. Chen, W. C. Lee, and M. S. Chen, "On social-temporal group query with acquaintance constraint," *Proceedings of the Vldb Endowment*, vol. 4, no. 6, 2011.

[3] D. N. Yang, C. Y. Shen, W. C. Lee, and M. S. Chen, "On socio-spatial group query for location-based social networks," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 949–957.

[4] W. Liu, W. Sun, C. Chen, Y. Huang, Y. Jing, and K. Chen, "Circle of friend query in geo-social networks," in *International Conference on Database Systems for Advanced Applications*, 2012, pp. 126–137.

[5] K. Stefanidis and G. Koloniari, "Enabling social search in time through graphs," in *Proceedings of the 5th International Workshop on Web-scale Knowledge Representation Retrieval & Reasoning*, 2014, pp. 59–62.

[6] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 656–665.

[7] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer, "An asymptotically optimal multiversion b-tree," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 5, no. 4, pp. 264–275, 1996.