Temporal Social Network: Storage, Indexing and Query Processing*

Xiaoying Chen[1], Chong Zhang[2], Bin Ge[3], Weidong Xiao[4]

*Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology, Changsha 410073, China
+Collaborative Innovation Center of Geospatial Technology, China

{[1]chenxiaoying1991, [2]leocheung8286}@yahoo.com
[3]gebin1978@gmail.com, [4]wilsonshaw@vip.sina.com

ABSTRACT

With the increasing of requirements from many aspects, various queries and analyses arise focusing on social network. Queries like finding users, friends or social activities satisfying a certain period gives temporal insights into retrieval or statistics, hence augmenting temporal query capability in such context, temporal social network (TSN), is meaningful. We propose three kinds temporal queries in social network, aiming to explore temporal dimension in users, relationships and social activities. A storage model is designed to logically and physically represent TSN, and then we propose two index structures, TUR-tree and TUA-tree for accelerating query process. Query processing algorithms are designed for the three queries, and we evaluate our idea on a dataset which is synthetically generated from real dataset, and experimental results show that our indexes and query processing are effective and scalable.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - *query processing*

General Terms

Algorithms, Measurement, Performance

Keywords

Social Network, Temporal, Storage, Index, Query Processing

1. INTRODUCTION

Online social networks have become more and more popular, numerous sites such as Facebook, Twitter and LinkedIn allow users to interact and share content using social links. Based on vast amount of information contained in social networks, personalized and expressive search features are essential for users, vendors or third-parties based on different purposes. In various kinds of data, time is a common and necessary dimension in social networks. For instance, when a user logins or logouts the community system, there is often a time stamp recording such action. When two users become friends, a time stamp also records this event, and similar for unfriending case. Similarly, when a user posts text, photo, video, or comments others' posts, or shares web links, these social activities are all recorded with time stamps.

When these temporal information is taken into consideration in an ordinary query for social networks, it becomes interesting to discover that some historical insights are associated with the results, e.g., find some users who are active during a certain period, or find pairs of friends valid in some year, or find some activities posted at some time. Such queries add temporal axes for user requirement, which is able to distinguish *old* and *new*, or *active* and *inactive* users, friends or activities. However, such kind of query functions are not studied well in academic community, nor developed in industrial platforms.

In this paper, our goal is to explore temporal queries in social networks associated with time dimension, we call it temporal social networks (TSN). And we focus on the following three kinds of queries.

(1) **Friends of Interesting Activities (FIA) query**. FIA queries aim to find a user's friends who are involved in some given social activities. For instance, George wants to find which friends of him post the texts containing *coffee* and *pasta* during last two weeks. This query would help users to find interesting events or people along the time axis.

(2) **Users of Time Filter (UTF) query**. UTF queries aim to find certain users who are not only active during a given time interval, i.e., the period of logon status intersects with the given time interval, but also whose friends take part in some given interesting activities. For example, a fashion advertiser plans to look for some users who are active in recent month, and the user's friends have taken part in the social activities containing *boot*, because the advertiser wants to persuade the users to buy their products utilizing influence from the friendships.

(3) **Group of Users with Relationship Duration (GURD) query**. GURD queries aim to find a set of groups, where the number of users in each group is equal to a given

*This work is supported by NSF of China grant 61303062 and 71331008.

number, and average intimate degree of it satisfies a given value (here average intimate degree is measured by average relationship duration, relationship duration is calculated by difference between current time and friend-make time), and all the members of it have taken part in some given activities. For instance, a pizza restaurant wants to send coupons to groups of users for promotion, so it aims to find four-member groups (due to the size of its dining table), and the average relationship duration in each group is not less than one year, and all four members have participated in the activities containing *pizza*.

Motivated by the important role time may assume in social query, our goal is to design appropriate storage scheme, index for typical temporal-social query, which has been mostly ignored so far, to improve the expressiveness and quality of social search queries. We first formally describe the problem, and use a storage model to store TSN data, then we design two indexing structure to accelerate the query processing. We implement our indexes and algorithms and empirically verify our methods are feasible. Our main contributions are listed as following:

- Three new query patterns are proposed in social network, taking time dimension into consideration.
- A storage model for TSN is presented, including logical view and physical model.
- Querying processing algorithms are designed, along with two indexing structures to solved the three query patterns.

The rest of this paper is organized as follows. Problem is formally defined in section 2. In section 3, storage model for TSN is presented. Two indexing structures are described in section 4, followed by query processing in section 5. In section 6, we carry out our experiments, and related works are surveyed in section 7. Finally, section 8 concludes the paper with directions for future works.

2. PROBLEM DEFINITION

In this paper, we focus on the social network of undirected graph, however, it is straightforward to extend it to directed graph. Our model is formally defined as following:

Given an undirected graph G=(V, E), each vertex v_i in V represents a user in the community and is associated with a time interval [vt_s, vt_e), meaning the valid period during which v_i exists (or being logon status) in community, and [vt_s, $*$) means v_i is still in the community now. Each edge (v_i, v_j) in E represents a friend relationship between user v_i and v_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. Similarly, [et_s, $*$) means the relationship still exists at current.

For a given social network graph G, there is also a set A of activities, 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. Each v_i in G can connect to an activity a_k in A, which means that v_i gets involved in a_k, we use term *participation* to describe the 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. 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. Assuming user v_i publishes a post a_k in the Facebook at time t_p, and then her/his friend v_j comments the post at some time t_q, thus the two participating actions could be formally represented by (v_i, a_k, t_p) and (v_j, a_k, t_r), respectively.

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 v_1 logins community at time vt_1 and does not logout at current, and user v_1 and v_2 become friends at et_1 and unfriend each other at et_3, and v_4, v_7 and v_2 participate in a_2 at t_4, t_5 and t_6, respectively.

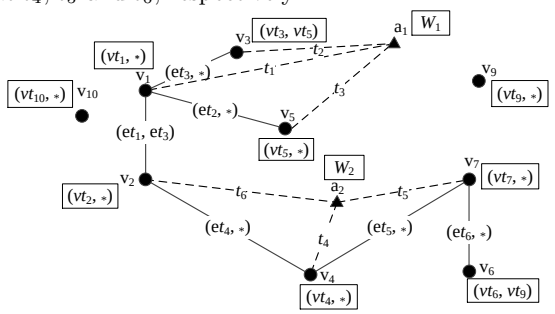


Figure 1: Temporal Social Network Example

3. STORAGE MODEL

Based on the description of temporal social network (TSN), we give TSN's storage model, in which two perspectives are presented: logical view and physical model.

3.1 Logical View

In logical view, users and activities are partitioned horizontally, and social events are partitioned vertically along the time axis. In the upper half of the logical view, along the time axis, the actions which a specific user makes are presented in chronological order. Further, in the lower half, a list of users which participate in a specific activity also can be obtained along the time axis. Figure 2 shows an example of the logical view, along the time axis, user v_1 logins social network at time t_1, makes friends with user v_2 at time t_2 and participates in activity a_1 at time t_3. And activity a_1 is participated in by user v_1 and v_2 at time t_3 and t_4, respectively.

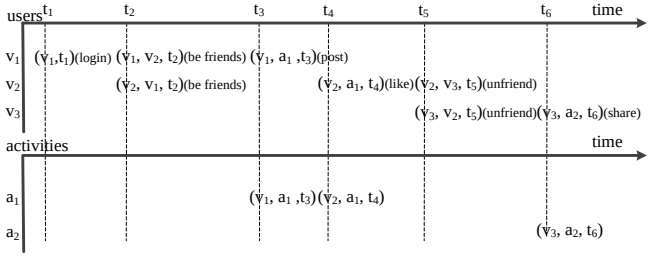


Figure 2: Logical View Example

3.2 Physical Model

To efficiently access temporal social network, we design a

physical model for storing TSN. The user and activity information is separately stored, and users are clustered into pages on disk, which is similar for activities. In particular, each user page consists of several items, where each item represents a user, including user id, name and etc, to look up quickly, it also contains three pointers, the first one references to a list of time intervals, each of which represents a pair of login and logout time stamps of the user, and the second one points to a list, each item of which contains the user's friend id, besides friend-making time and unfriending time, and the third one also points to a list, each item of which contains activity id and corresponding participating time related to the user.

The model for activity is similar to user's, i.e., each item in activity page contains activity id, the link to keywords and other objects in the activity, and a pointer references to a list, where each item represents the user participating in it as well as the time stamp. Additionally, the activity in user page also points to the corresponding item in activity page. Note that, the lists in the physical model do not belong to user page or activity page, they are sequentially stored on disk. Figure 3 illustrates a physical model example.

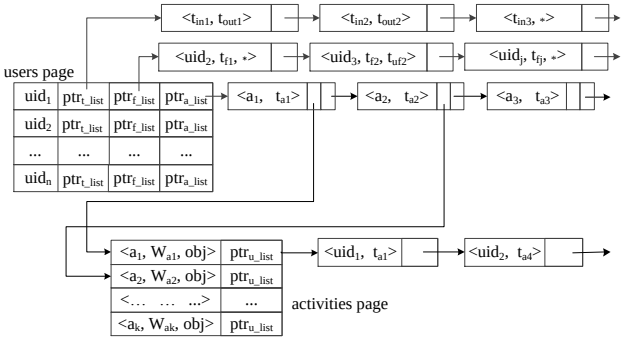


Figure 3: Physical Model Example

4. INDEXES

The storage model works for storing temporal social network, when processing queries, the sequential scanning would be inefficient under large volume. Thus it is essential to develop indexing technique for accelerating query processing. However, it is difficult to build one almighty index structure to cover users, relationships and activities with temporal information. We design 2 structures, one is called Temporal Users and Relationships tree (TUR-tree), for temporally indexing users and relationships, the other is called Temporal Users and Activities tree (TUA-tree), for users and their activities.

4.1 TUR-tree

Considering that both users and relationships are associated with time interval, and queries are subject to time range type, thus, we propose to use MVB-tree[1] to index users and relationships. Our main contribution is to adapt temporal users and relationship to MVB-tree.

In MVB-tree, the entry of leaf node is formatted as $<key, t_s, t_e>$, where key is the value to index, and $[t_s, t_e)$ is the valid time interval for key. On the other hand, the entry of non-leaf node is formatted as $<key, t_s, t_e, subnodeid>$, where the function of key is for comparing and routing, and $subnodeid$ points to the root of subtree.

In our scenario, user id is usually used as key to query, so we can directly use user id as the key in MVB-tree. However, relationship is different from user id, and can't be applied to MVB-tree straightforward. Hence, we use encoding technique to generate keys for MVB-tree. In particular, we use string concatenating to generate keys both for users and relationships. To generate key for a user v_i identified by uid_i, we combine 0 and uid_i, resulting $0|uid_i$ (| represents concatenating operator), as the key in MVB-tree. For relationship of uid_i and uid_j, we use 1 as prefix, and concatenate uid_i and uid_j, resulting $1|uid_i|uid_j$ and $1|uid_j|uid_i$, note that, for undirected graph, we generate two keys for one relationship. Additionally, for fast looking up relationships when given a user, links are made among user leaf node entry and its corresponding relationship leaf nodes.

Figure 4 depicts an example for construction of TUR-tree. It illustrates the process that three users joins social network and then make friends. Assuming that the length of user id is three, and it is only composed of 0-9 and a-z (this assumption is also used in the rest of the paper). In particular, three users with identifiers 001, 002 and 003 joins the network at time 1, 2 and 4, respectively, and 001 and 002 make friends at time 3, and 002 make friends with 003 at time 5. In this example, we set the capacity of a node is 3, and Figure 4(a) presents the result structure of TUR-tree after 001 and 002 join the network, i.e., there is only one node, namely node A. After 001 and 002 make friends, two entries representing the relationship are inserted into node A, which causes overflow of A, and according to the split rule of MVB-tree, a version split should be carried out, i.e., copying the current version of entries in node A, and combining them with the new entries to form a new node, if it is overflow, key split is carried out, after that for building links between user leaf node entry and its relation leaf nodes, entry <0|001, 1, *> points to the node containing <1|001|002, 3, *>, and similar for other user leaf node entries, thus the above sequential operations generate a structure in Figure 4(b). In node B, the two entries are copied from node A, while for node A, root R_1's first entry refers to it with time interval [1, 3). At time 4, user 003 joins the network, resulting insertion to node B, and Figure 4(c) describes the structure. At time 5, user 002 make friends with 003, two new entries are inserted, similar to the process in Figure 4(b), this causes node C overflow, and such overflow propagates to root R_1, similar to leaf node split, R_1 is split into R_1 and R_2, and Figure 4 shows the result.

4.2 TUA-tree

The temporal information associated with activities is different from that with users and relationships, i.e., when user participates an activity, the temporal type is time stamp not interval, hence it is not necessary to use MVB-tree. Another fact is for querying activities, activity identifiers are not usually used as searching keys, but user ids, time predicates and keywords. Thus, TUA-tree is designed to cover user id, time and keywords, in fact it is a hybrid structure of B^+-tree and Bloom Filter[6]. We give the detail structure of TUA-tree as following.

TUA-tree is a B^+-tree-like structure, the entry of leaf node is formatted as $<key, ptr>$, where ptr points to an activity, and key is concatenation of user id and time associated with the activity. Node split, merge and redistribution operations are carried out according to key. The internal node structure is similar to that of B^+-tree, i.e., entries are distinguished

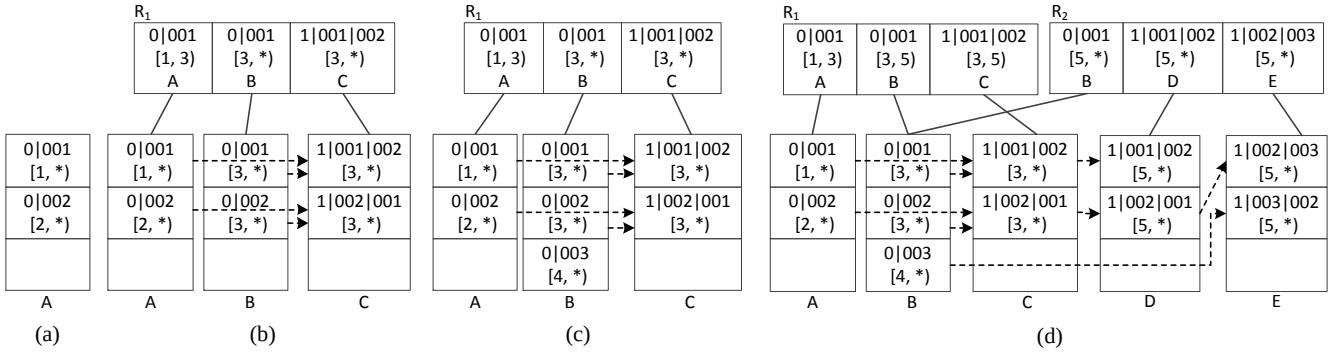


Figure 4: An Example of TUR-tree

by router and pointer, and the number of pointer entries is larger than that of router entries by one, different from B$^+$-tree, pointer entry also contains Bloom Filter of all keywords in the referred subtree.

Figure 5 illustrates a process of TUA-tree construction, and the capacity of tree node is set to 2. First, user 001 participates in activity a_1 at time 1, then 002 get involved in a_1 at time 2, resulting a leaf node showed in Figure 5(a). Then user 003 publishes activity a_2 at time 3, which causes overflow, and node split operation follows, resulting a new root linking two leaf nodes, we can see from Figure 5(b), the left pointer entry in the root node contains Bloom Filters BF_1 which is calculated from the keywords from a_1, similarly, BF_2 is calculated from the keywords from a_2. After that, user 001 participates a_2 at time 4 (Figure 5(c)), and user 004 posts activity a_3 at time 5 (Figure 5(d)). At last, user 005 and 001 participates in a_4 at time 6 and 7, respectively, resulting the root node split and tree level increased, and BF_6 contains keywords from a_1, a_2 and a_4, while BF_7 contains keywords from a_1, a_2, a_3 and a_4.

5. QUERY PROCESSING

In this section, we present algorithms for processing FIA query, UTF query and GURD query.

5.1 FIA Query

Given a user v_q, a keyword set W_q, and a time interval $[t_s, t_e]$, a FIA query (v_q, W_q, t_s, t_e) aims to find a set of pairs $\{<v_k, la_k>\}$, in each of which v_k is v_q's friend valid during the period $[t_s, t_e]$, la_k is a list of activities participated in by v_k during $[t_s, t_e]$, and each activity in la_k overlaps W_q (i.e., the intersection of activity's keyword set and W_q is not null).

For processing a FIA query (v_q, W_q, t_s, t_e), an ideal approach is using join algorithm to search TUR-tree and TUA tree simultaneously, however such concept can not be realized by our structures, due to combination of user's and his friend's id representing relationship key in the index. We propose a two-phase processing algorithm, i.e., first TUR-tree is traversed to retrieve friends of v_q satisfying the temporal predicate, then these friends as well as (W_q, t_s, t_e) are used as query input to TUA-tree to look up final results.

In particular, for the first phase, a MVB-tree range query ($[1|v_q|000, 1|v_q|zzz], [t_s, t_e]$) is generated, and submitted to TUR-tree to retrieve v_q's friends, set F, valid between t_s and t_e. The first phase is a straightforward invocation of MVB-tree's searching method.

In the second phase, a query which consists of F and (W_q, t_s, t_e) is submitted to TUA-tree to retrieve final results. Note that, in such query, F is a set containing users, if TUA-tree is traversed at one time for each element in F, it would be costly. Hence, we propose a batch search algorithm for query F with (W_q, t_s, t_e) in TUA-tree. Initially, the root of TUA-tree is loaded into memory, including router entries (re) and pointer entries (pe), formatted as $root$=<pe_0, re_1, pe_1, re_2. pe_2, ..., re_n, pe_n>. For batch comparison, we generate an interval $intvl$=$[f_{min}|t_s, f_{max}|t_e]$, where f_{min} and f_{max} are minimum and maximum user id in F, respectively. After that, for each pe_i, if $intvl$ intersects with (re_i, re_{i+1}] (note that, for the first pe, pe_0, the interval should be ($-\infty$, re_1], and (re_n, $+\infty$) for the last one, for simplicity, we omit the judgment in the algorithm presentation) as well as Bloom Filter in pe_i intersects with W_q, the child node referenced by pe_i is inspected by query (F_{sub}, W_q, t_s, t_e), where F_{sub} is the intersection of F and the users set in $[re_{i-1}, re_{i+1}]$. When leaf node of TUA-tree is reached, each entry is examined whether it satisfy current query predicate, and the results are fetched into list.

Algorithm 1 presents the second phase of FIA query. Tree root is loaded in line 1, and then from line 3 to line 19, function $FIA2nd\text{-}P()$ recursively traverses the tree, with continuously dividing F into subset. If the parameter $node$ is an internal node (line 4), variable $intvl$ is generated (line 5). From line 6 to 11, each pointer entry of the node is inspected to detect whether its child node satisfy the predicate, if it does, only the users who are contained by the interval $[re_i, re_{i+1}]$ are preserved to descend the subtree (line 8 and 9). Otherwise, i.e., $node$ is a leaf (line 12), each entry of the leaf is inspected (line 13 and 14), and the result is added into $Rlist$ (line 15).

FIA query $2nd\text{-}P$ processing example. We use Figure 5 (e) as a running example, and assuming query is (001, 002, $a_2.W_{a_2}$, 3, 5) and the keyword sets of each pair activities are not overlapped. First, root node <pe_0, 001|7, pe_1> is loaded, then for pe_0, $[001|3, 002|5]\cap(-\infty, 001|7]\neq\phi$ and BF_6 contains keywords in a_2, hence the subtree referenced by pe_0 is traversed, then similarly for pe_1, the subtree of which is also traversed. Then node A and B is loaded, the similar processing is carried out until descending the leaf node entries <001|1, Ptr_{a_1}>, <001|4, Ptr_{a_2}>, <002|2, Ptr_{a_1}>, <003|3, Ptr_{a_2}>, and then by function $compare()$, <001|4, Ptr_{a_2}> is the result.

5.2 UTF Query

Given a keyword set W_q, and a time interval $[t_s, t_e]$, a

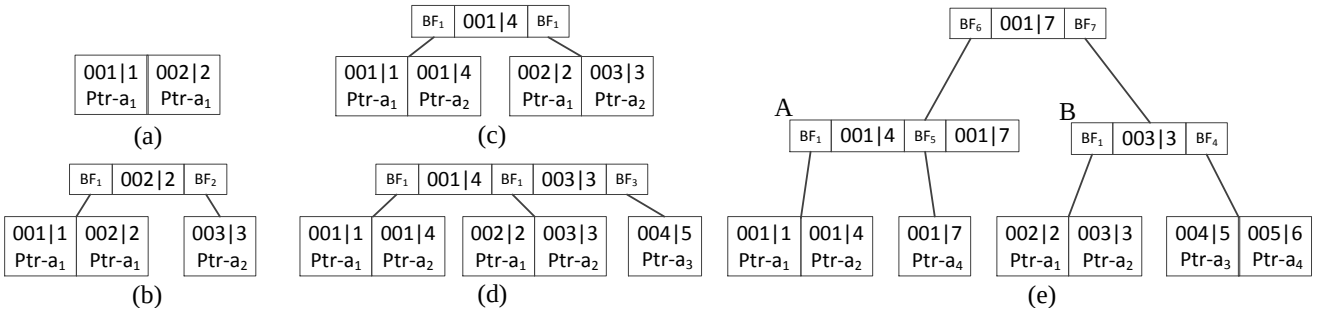


Figure 5: An Example of TUA-tree

Algorithm 1 FIA Query 2*nd-P* Processing

Input: q=(F, W_q, t_s, t_e)
Output: *Rlist*
1: $root$=<pe_0, re_1, pe_1, re_2, pe_2, ..., re_n, pe_n>
2: $node$=$root$
3: $FIA2nd\text{-}P(node, F, W_q, t_s, t_e)$
4: **if** *node* is non-leaf **then**
5: $intvl$=$generateIntvl(F, t_s, t_e)$
6: **for** each $pe_i \in node$ **do**
7: **if** $((re_i, re_{i+1}] \cap intvl \neq \phi) \wedge (pe_i.BF.test(W_q)$ ==true) **then**
8: F_{sub}=$calcIntersection(F, re_i, re_{i+1})$
9: $FIA2nd\text{-}P(pe_i.childNode, F_{sub}, W_q, t_s, t_e)$
10: **end if**
11: **end for**
12: **else**
13: **for** each $entry_i \in node$ **do**
14: **if** $entry_i.compare(F, W_q, t_s, t_e)$==true **then**
15: $Rlist \leftarrow entry_i$
16: **end if**
17: **end for**
18: **end if**
19: **return** *Rlist*

UTF query (W_q, t_s, t_e) aims to find a set of pairs {<v_k, FA_k>}, in each of which v_k is the user existing in the social network during [t_s, t_e], and FA_k is a set {<v_j^k, la_j^k>} similar to the results of FIA query, i.e., v_j^k is v_k's friend, and la_j^k is a list of activities participated in by v_j^k during [t_s, t_e] as well as satisfying overlapping W_q.

The aim of previous query, FIA query, is to find a list of friends of the given user, while for UTF query, the aim is different, which is to find a set of users active during a given time period, whose friends take part in some certain activities. The processing for UTF query is a little similar to that of FIA query, i.e., there are also two phases, first is using TUR-tree, then TUA-tree is traversed.

In the first phase, a query formatted as ([0|000, 0|zzz], [t_s, t_e]) is submitted to TUR-tree, using range query algorithm of MVB-tree, a set of leaf node entries which contain the satisfied users are retrieved, then following the links to leaf node of the corresponding relationships, a set of friends are fetched, specifically, for each result user v_i, a list friends F_i of v_i is formed.

In the second phase, a query formatted as ($\bigcup_i(F_i)$, W_q, t_s, t_e) is submitted to TUA-tree, similar to 2*nd-P* in FIA query, batch query is executed and a set F_{2nd} of friends and corresponding activities are found. At this point, for each v_i found in the first phase, we check whether his friend list F_i fetched in the first phase intersects with the friend set F_{2nd} in the second phase, i.e., F_{int_i}=$F_i \cap F_{2nd}$, if F_{int_i} is not null, then v_i and F_{int_i} as well as the corresponding activities are collected as results.

Algorithm 2 UTF Query Processing

Input: q=(W_q, t_s, t_e)
Output: *Rlist*
1: LE=$TURRangeQuery$([0|min, 0|max], [t_s, t_e])
2: {<v_i, F_i>}=$fetchRelations(LE)$
3: F_{2nd}=$2nd\text{-}P(\bigcup_i(F_i), W_q, t_s, t_e)$
4: **for** each F_i **do**
5: F_{int_i}=$F_i \cap F_{2nd}$
6: **if** $F_{int_i} \neq \phi$ **then**
7: $Rlist \leftarrow$<v_i, F_{int_i}>
8: **end if**
9: **end for**
10: **return** *Rlist*

Algorithm 2 presents the processing of UTF query. In line 1, function $TURRangeQuery$ retrieves the users who are active between t_s and t_e, and then through links, the corresponding friends are fetched in line 2. Next, similar to the second phase in FIA, in line 3, set $\bigcup_i(F_i)$ is refined by condition (W_q, t_s, t_e) using function 2*nd-P*. From line 4 to the end, for each F_i, intersection examination is made to refine and then collect the results.

UTF query example. We use Figure 4 (d) and 5 (e) as a running example, and assuming query input is ($a_1.W_{a_1}$, 1, 2). In the first phase, through the TUR-tree, we find leaf node <0|001, 1, *, Ptr_{001}>, <0|002, 2, *, Ptr_{002}> satisfying time condition [1, 2]. Then using Ptr_{001}, the set of friends F_{001}={002} of user 001 is found, similarly, F_{002}={001, 003} is returned. At the second phase, firstly a total friends set {001, 002, 003} is formed, then using algorithm 2 with input ({001, 002, 003}, $a_1.W_{a_1}$, 1, 2), F_{2nd}={001, 002} is returned. Then because $F_{001} \cap F_{2nd} \neq \phi$ and $F_{002} \cap F_{2nd} \neq \phi$, the result is {<001, {<002, a_1>}>, <002, {<001, a_1>}>}.

5.3 GURD Query

Given a number m, a time length (duration) t_d, and a keyword set W_q, a GURD query (m, t_d, W_q) aims to find groups of users, in each group of which users form a connected graph, and the number of users is m, and average relationship duration (ARD) is not less than t_d, and for each user in the group, at least one activity participated in overlaps W_q. Here, ARD is defined as following: assuming the number of users in a group is n, and let $d(i, j)$ be the relationship duration between user v_i and v_j in the group (current time minus relationship establishing time, note that

this query implies all the connections in a group are valid at current), if there is no relationship between v_i and v_j, $d(i,j)$=0, then

$$ARD = \frac{\sum_{i \neq j}^{n} d(i,j)}{n(n-1)} \quad (1)$$

Note that, in equation (1), for v_i and v_j, both $d(i,j)$ and $d(j,i)$ are taken into consideration.

The aim of GURD query is to find groups of users, which is totally different from the previous two queries. In this query processing, TUA-tree is mainly used to accelerate the process, this is because calculating user group is costly using TUR-tree. The basic work flow is first using TUA-tree to find a set U_c of users who participate in the activities constrained by W_q, then we propose a group forming algorithm to construct satisfied groups iteratively. The goal of the algorithm is to terminate the iteration as early as possible.

In particular, W_q is submitted to TUA-tree, at this time, only Bloom Filter is used to route the query descending the candidate subtree, resulting U_c, each user of which satisfies the activity condition. Next is our group forming algorithm. We first calculate relationship duration between each pair of users in U_c if they are friends, such step could be accomplished by searching TUR-tree or directly looking up the storage model. Then we sort the relationship duration of each pair of friends in U_c in descending order, resulting RD= {<(v_i, v_j), $d^k_{i,j}$> | v_i, $v_j \in U_c \wedge v_i$ and v_j are friends $\wedge$ $1 \leq k \leq |RD|$}, where $d^k_{i,j}$ is the relationship duration between v_i and v_j, and is the kth one in descending order of $|RD|$ durations. Next, top element is constantly removed from RD, and we compare the following two variables: R_{um_k}=$\sum_{n=k}^{k+((m-1)m/2)} d^n_{i,j}$ and R_{mrs}=$t_d \times m(m-1)/2$, if $R_{um_k} \geq R_{mrs}$, it implies possibility for finding groups satisfying the ARD requirement, and then v_i and v_j are used to find their friends from the rest users of RD, and for each possible connected graph containing v_i and v_j with vertex amount being m, ARD is calculated and only the group with $ARD \geq t_d$ is collected into result list. Otherwise, i.e., $R_{um_k} < R_{mrs}$, which means it is impossible for this edge to find other connected vertices to form an m member group whose ARD is not less than t_d, hence, the rest elements in RD are discarded, and iteration is terminated.

Algorithm 3 GURD Query Processing

```
Input:  q=(m, t_d, W_q)
Output: Rlist
 1: U_c=searchTUAtree(W_q)
 2: FD=sortFD(U_c)
 3: while FD ≠ φ do
 4:    FD =FD \ {<(v_i, v_j), d^k_{i,j}>}
 5:    if R_{um_k} < R_{mrs} then
 6:       break
 7:    else
 8:       G_{i,j,m}=constructGroups(v_i, v_j, FD, m)
 9:       for each g ∈ G_{i,j,m} do
10:          if g.ARD ≥ t_d then
11:             Rlist←g
12:          end if
13:       end for
14:    end if
15: end while
```

Algorithm 3 presents GURD query processing. In line 1, W_q is searched in TUA-tree to find qualified users, which are organized as friend pairs sorted by their relationship duration (line 2). From line 3, top element in FD is constantly removed (line 4) and R_{um_k} is calculated, if $R_{um_k} < R_{mrs}$, iteration is terminated (line 6), otherwise, function *constructGroups* finds a set $G_{i,j,m}$ of connected graphs, each of which contains m users including v_i and v_j (line 8), and each group in $G_{i,j,m}$ is inspected (line 9) and the one with $ARD \geq t_d$ is added into *Rlist* (line 10 and 11).

Group forming example for GURD query. We use Figure 6 as a running example. Assuming U_c={$v_1, v_2, ..., v_{10}$}, m=3 and t_d=5, hence R_{mrs}=15. After sorting, FD={<(v_9, v_{10}), 9>, ... ,<(v_5, v_8), 1>}. In iteration 1, <(v_9, v_{10}), 9> is removed from FD, due to R_{um_1}=9+8+7=24 > R_{mrs}, function *constructGroups* is invoked, resulting $G_{9,10,3}$={(v_5, v_9, v_{10})}, and ARD of (v_5, v_9, v_{10}) is (3+9)/3<5, so the group is discarded. Similarly, in iteration 2, due to R_{um_2}=22 > R_{mrs}, so $G_{1,5,3}$={(v_1, v_2, v_5), (v_1, v_3, v_5), (v_1, v_5, v_8), (v_1, v_5, v_{10})}, and at this time *Rlist*={(v_1, v_2, v_5)}. At iteration 5, due to R_{um_5}=14 < R_{mrs}, the iteration is terminated and final result *Rlist*={(v_1, v_2, v_5)} is returned.

6. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate our three queries on a hybrid dataset, where *hybrid* means that, we download a real dataset describing YouTube social network from KONECT[1], however, this dataset only contains edges and their time stamps (friend-making time), hence, base on this dataset, we synthetically generate user login and logout time, activities and participation time. In particular, for each user, 0 to 10 login and logout pairs are randomly generated, and edges are randomly picked up to be assigned an unfriending time stamp (this is because there are already friend-making time stamps in the original dataset). Then we generate 5 million activities containing different keyword sets, and we assign users to activities in a Zipf distribution, and randomly assign them participating time stamps. Table 1 describes our dataset, the first column means the amount of the items with temporal dimension, the second column means the amount without temporal dimension.

Table 1: Dataset Description

	number of records	cardinalty
users	12.78 million	3.32 million
relationships	14.33 million	5.45 million
activities	30.23 million	5 million

We implement TUR-tree and TUA-tree as well as query processing algorithms in Java. To make comparison, we use non-index query processing as a baseline, i.e., for the three kinds queries, the storage model is directly accessed and sequential scans are carried out. And then we use our two indexes to process the three queries according to the approaches in section 5. 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 processor, 8GB memory and 500GB disk. Table 2 describes the query parameters.

6.1 FIA Query

In this experiment, first, we vary user's degree, and Figure 7(a) shows the results. Response time increases with

[1]http://konect.uni-koblenz.de/networks/youtube-u-growth

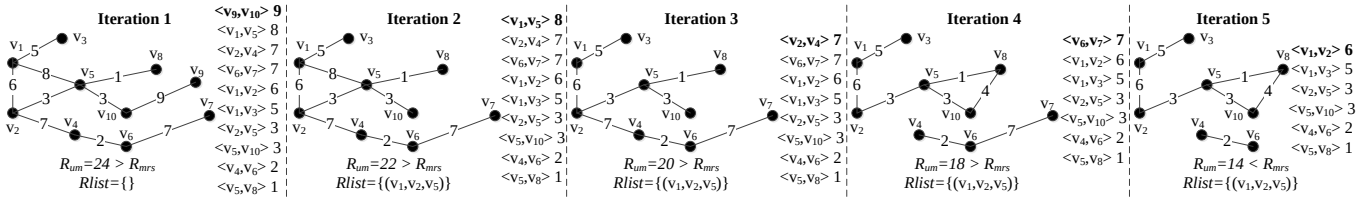


Figure 6: Group Forming Example in GURD Query

Table 2: Parameters in Experiment

queries	parameters	domain	default
FIA	v_q's degree	10 - 100	15
	W_q's cardinality	1 - 5	3
	$length_{[t_s,t_e]}$/temporal extent	0.1% - 3%	1%
UTF	W_q's cardinality	1 - 5	3
	$length_{[t_s,t_e]}$/temporal extent	0.1%-3%	1%
GURD	m	3 - 9	5
	t_d	1 - 5	3
	W_q's cardinality	1 - 5	3

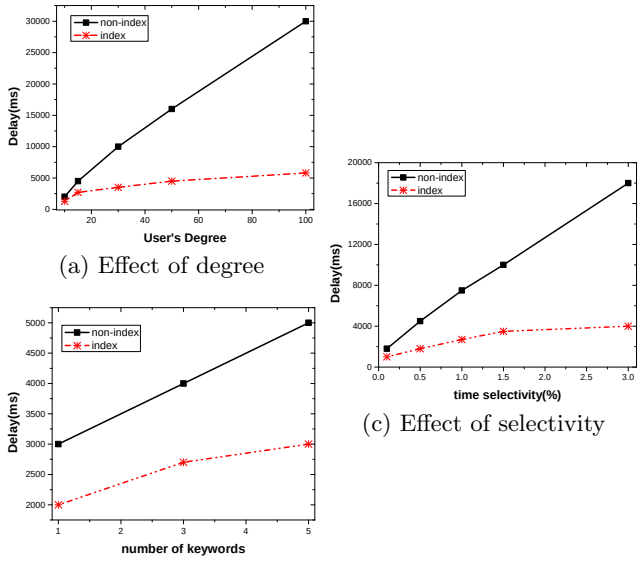
degree for both non-index and index approaches. This is because a user with a large degree (meaning he has lots of friends) would involve a wide search in TUR-tree and then in TUA-tree. However, we can see the performance of non-index degenerates quickly, while our two indexes accelerate the processing apparently, above 50% order of magnitude. Specifically, response time for non-index increases linearly, while it is logarithmic for the index one, due to the fact TUR-tree adapts MVB-tree's architecture which is asymptotic to the performance of B-tree, and TUA-tree is similar to B-tree. Next, Figure 7(b) shows the results on varying the number of querying keywords, we can see the larger the number of keywords is, the more responding time it takes, due to more comparisons would happen when sequential scanning on disk or traversing TUA-tree. At this time, we can see the change of response time is slower than that in Figure 7(a), this can be explained that the keywords have less impact than user's degree to query performance. Figure 7(c) presents the results on varying time selectivity ($length_{[t_s,t_e]}$/temporal extent), still we can see our indexes accelerate processing logarithmically and have a good scalability.

6.2 UTF Query

Next, similar to FIA query experiment, we vary number of keywords and time selectivity for testing UTF query performance. From Figure 8, we can see processing time of UTF query is larger than that of FIA query, this is because there is no given user in UTF query, which causes a large cardinality query for searching storage model or TUR-tree, more items would be compared. However, we can see TUR-tree and TUA-tree greatly accelerate query processing.

6.3 GURD Query

In the following, we vary the number m of group members, Figure 9(a) shows that response time increases with m on both methods, this is because a larger m involves more combinations to be examined, however, we can see TUA-tree and group forming algorithm is effective for the premium performance they show. Then we vary the number of keywords, and results are similar to that of previous two queries (see Figure 9(b)). Figure 9(c) shows the results when increasing

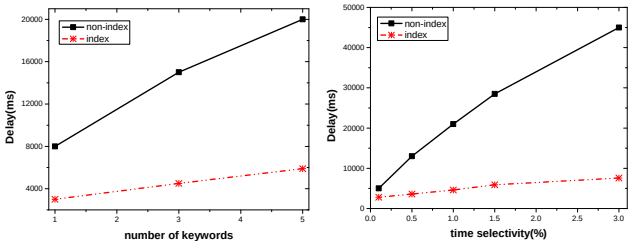


(a) Effect of degree

(b) Effect of keywords

(c) Effect of selectivity

Figure 7: Results of FIA Queries



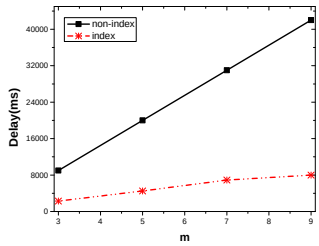
(a) Effect of keywords (b) Effect of selectivity

Figure 8: Results of UTF Queries

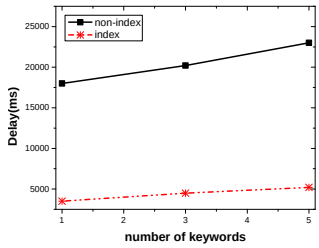
t_d, we can see at this time, response time decreases as t_d increases, this is due to our group forming algorithm, a larger t_d will terminates the iterations earlier.

7. RELATED WORK

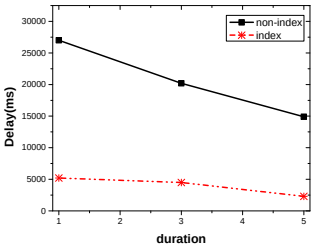
In recent years, plenty of works have focused on efficient methods for managing and querying social networks. One branch of works on social networks focus on efficient algorithms and data structures for searching users and friends of interest. However, these studies focus on evolution of graphs and topological characteristics of social network, ignoring examining how the social network links are being used by users to interact. Hence, some researchers propose activity network, a network that is based on the actual interaction between users rather than mere social network, and argue that this is more suitable to social network-based applications. B. Viswanath et al. study the activity network from



(a) Effect of m



(b) Effect of keywords



(c) Effect of t_d

Figure 9: Results of GURD Queries

Facebook New Orleans network[9], and observe that the activity network is rapidly changing over time, but many of the global structural properties remain relatively constant.

P. Holme et al. [3] present the emergent field of temporal networks, and discuss methods for analyzing topological and temporal structure, and models for elucidating their relations to the behaviors of dynamical systems. However, they only discuss some models proposed for temporal social networks and epidemiological contact networks without query processing method.

In [5], the authors present a model for capturing graph evolution through time based on snapshots and deltas, and introduce a general two-phase query plan based on snapshot reconstruction to evaluate any historical query. However, this work dose not provide efficient techniques and index structures for various types of queries.

U. Khurana et al.[4] build a graph data management system that focuses on optimizing snapshot retrieval queries over historical traces, and on supporting rich temporal analysis of large networks. Another storage approach was proposed in [7], namely, the historical evolving graph sequence (EGS). Various snapshots and deltas are explicitly stored, with temporally close snapshots being clustered together. Nevertheless, if the query asks for a single time point or a time interval (i.e., not the whole history), this approach will be ineffective.

In [8], a storage model maintains the current graph and deltas to previous time snapshots; as a result, the first step of evaluating a historical query is to reconstruct the corresponding snapshot or snapshots that relate to the query's temporal predicate. However such a reconstruction phase is costly, and this is an issue also in other related works[5, 4]. Chronos[2] is a storage and execution engine designed and optimized specifically for running in-memory iterative graph computation on temporal graphs, which takes further in locality, parallelism, and incremental computation.

However, in work[5, 4, 7, 2], the activity network is not considered. The work[8] defines two kinds of entities in graph: users and objects. In our paper, we combine user social network and activity network, which consists of two kinds of entities: users and activities, and various temporal social queries can be satisfied.

8. CONCLUSIONS

In this paper, we argue that time axis in social network is an important and useful tool to give insight into retrieval or statistics, and augmenting temporal query capability in such context is meaningful. We propose three kinds queries, namely FIA, UTF and GURD query, and model the temporal social network (TSN) with users, relationship and activity as well as corresponding temporal labels. A storage model is designed to logically and physically represent TSN, and then we propose two index structures, TUR-tree and TUA-tree for accelerating query process. We propose algorithms of query processing for the three queries, and evaluate our idea on a dataset which is synthetically generated from real dataset, and experiment results show that our indexes and query processing are effective and scalable. In the future, we plan to extend this work to geo-location application, and solve spatio-temporal queries in social networks.

9. ACKNOWLEDGMENTS

This work is supported by NSF of China grant 61303062 and 71331008. We would like to thank Prof. Dai and Dr. Hu for helping with the proof.

10. REFERENCES

[1] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion b-tree. *The VLDB Journal*, 5(4):264–275, Dec. 1996.

[2] W. Han, Y. Miao, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, W. Chen, and E. Chen. Chronos: A graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 1:1–1:14, New York, NY, USA, 2014. ACM.

[3] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519(3):97 – 125, 2012. Temporal Networks.

[4] U. Khurana and A. Deshpande. Efficient snapshot retrieval over historical graph data. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 997–1008, April 2013.

[5] G. Koloniari, D. Souravlias, and E. Pitoura. On graph deltas for historical queries. *CoRR*, abs/1302.5549, 2013.

[6] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):124–141, Jan 2001.

[7] C. Ren, E. Lo, B. Kao, X. Zhu, and R. Cheng. On querying historical evolving graph sequences. *PVLDB*, 4(11):726–737, 2011.

[8] 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*, Web-KR '14, pages 59–62, New York, NY, USA, 2014. ACM.

[9] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2Nd ACM Workshop on Online Social Networks*, WOSN '09, pages 37–42, New York, NY, USA, 2009. ACM.