Social Search Queries in Time

Georgia Koloniari
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
gkoloniari@uom.gr

Kostas Stefanidis
Institute of Computer Science
FORTH
Heraklion, Greece
kstef@ics.forth.com

ABSTRACT

Recently, social networks have attracted considerable attention. The huge volume of information contained in them, as well as their dynamic nature, makes the problem of searching social data challenging. In this position paper, to increase the effectiveness of social search queries, we propose exploiting the temporal information available in social networks. In particular, we introduce different types of queries aiming at satisfying information needs from different perspectives. We present a formal graph and query model augmented with time and outline methods for query processing and time-dependent ranking. Finally, we identify several directions for future work.

1. INTRODUCTION

Due to the increasing popularity of social networks and the vast amount of information contained in them, recently there have been many efforts in enhancing web search based on social data. This has lead to the emergence of social search that utilizes the underlying graph structure and the content of a social network to provide both more personalized and more expressive search features for the users.

An important dimension of social networks is their dynamic nature. New information is added through user activities and updates occur both in the structure of the graph and the content shared representing respective changes in the users interests. This temporal aspect of the information should be exploited and influence social search either explicitly by enabling users to query for particular time points or periods, or implicitly by providing the most recent results and higher ranking of fresher information [4, 3]. Motivated by the important role time may assume in web search, our goal is to exploit the temporal information hidden in a social network, which has been mostly ignored so far, to improve the expressiveness and quality of social search queries.

To deal with the temporal aspect of the social graph, two basic approaches have been followed. In the first approach, a log-like file recording the updates that occur in the social graph is maintained [8, 7, 6]. This log constitutes the delta between snapshots of the graph at different time points and, by applying appropriate portions of it, one can create any snapshot required. For a time-dependent query, one needs to construct the required snapshot(s) involved in the query and then process the query on them. An approach that avoids the cost of snapshot construction, is introduced for RDF data [2] and uses an annotated graph model that incorporates temporal information.

In our work, we adopt the second approach and define a social graph model in which each element has a label maintaining its temporal information. In particular, both nodes, representing users and objects in the network, and the edges between them, representing the social relationships, have a label that indicates their *valid time* as defined for relational databases [10]. Unlike the previous approaches that exploit time to support queries mainly on the evolution of the graph through time [8, 7, 6], we deal with social search and propose a new time-enhanced query model.

We present two general query types: *user-centric* and *system-centric* queries. User-centric queries offer a personalized search feature by exploiting the social relationships of a user. In particular, we discern between user-centric queries for friends, which aim at discovering a user's friends that share common interests as indicated by their connections to a set of objects, and user-centric queries for objects that require objects that have connections to a subset of a user's friends. On the other hand, system-centric queries provide a global search feature with many applications in online-shopping [9] and target-advertising [1], so as to select the best target group for a new product or the best products to promote to a given user. Similarly to the user-centric queries, we discern between queries that aim at discovering users in the network that share common interests as expressed through connections to common objects, or objects that may be of interest to a particular subset of users.

To enable queries to express time explicitly, we extend user- and system-centric queries to time-dependent queries by adding time as a hard constraint in their definition, so as to filter out irrelevant results. To process such queries, the labels of the nodes in the system are processed to check for valid items according to the constraint in the queries. Furthermore, we also enable the implicit use of time to enhance the results of a query by providing a time-dependent ranking, so that more recent or fresher results are returned first. The ranking process exploits the time of the user activities recorded in the labels of the edges in the graph, i.e., the time the connections between the users and the retrieved results

PersDB 2013

were established.

The rest of the paper is structured as follows. Section 2 defines our graph and query model and how both are enhanced with temporal information. Section 3 outlines methods for query processing and introduces time-dependent ranking. Finally, in Section 4, we discuss our plans for extending this preliminary model.

2. MODEL

Typically, the entities of a social network represent users and objects. In this section, we first define users and objects, and then present our graph and query model.

2.1 User and Object Descriptions

Let u_i be a *user* described by a set of predicates $\{a_{i_1}, \ldots, a_{i_p}\}$, where each a_{i_j}, $1 \leq j \leq p$, is of the form ($a_{i_j}.attribute = a_{i_j}.value$). For example, an attribute can be *name*, *education*, *occupation*, *gender* or *age* and a corresponding predicate can be "*name* = *Alice*". See, for instance, Figure 1 for a user description.

name	=	Alice
education	=	college graduate
occupation	=	educator
gender	=	female
age	=	34

Figure 1: User description example.

Similar to users, we assume that each *object* o_z is described by a set of predicates $\{x_{z_1}, \ldots, x_{z_r}\}$, where each x_{z_j}, $1 \leq j \leq r$, is of the form ($x_{z_j}.attribute = x_{z_j}.value$). Objects are limited to applications that users use, events that users organize and attend, pages that users create and follow, and photos. For example, the description of the event *PersDB* for 2013 is shown in Figure 2.

type	=	event
name	=	PersDB
description	=	VLDB Workshop
topic	=	Databases
location	=	Trento
date	=	August 30, 2013
start time	=	9.00
end time	=	16.00

Figure 2: Event description example.

2.2 Graph Model

We model a social network as an undirected graph, $G = (V, E)$. The set of nodes V corresponds to the entities that belong to the social network, while the set of edges E captures the relationships between the entities that belong to V.

We discern between two types of entities. First, we consider the type *user* that consists of the social network users, or particpants. The second type, *object*, includes all other entities in the social network that are not users. In particular, we limit the objects to include applications, events, pages and photos. Thus, the set of nodes V is defined as the union of U and O, $V = U \cup O$, where U is the set of users and O the set of objects of the social network.

An edge $(v_i, v_j) \in E$, if v_i and v_j correspond to users u_i, $u_j \in U$ respectively, captures the friendship between the corresponding users. Note that our model supports symmetrical social friendships between users, such in Facebook. If v_i corresponds to a user $u_i \in U$ and v_j corresponds to an object $o_j \in O$, the edge (v_i, v_j) declares that user u_i is connected to object o_j. This means that in the underlying social network, u_i either uses or participates in some way in object o_j, e.g., u_i uses the application represented by o_j, or he/she is attending the event o_j, following the page o_j, tagged in the photo o_j, and so on. We assume that an object cannot consume another object and thus, object to object edges are not included in our graph model.

In this paper, we consider extending the typical graph model with temporal information towards making social search time-dependent. We are inspired by a traditional temporal database that includes temporal aspects, such as the valid and transaction time of data items [10]. Specifically, the valid time denotes the time periods during which a data item is true, or valid, for the real world, while the transaction time is the time period during which an item is stored in the database. For the purposes of this position paper, we consider only the valid time.

Here, we consider an element, node or edge, of a graph G as *valid* for the time period for which the corresponding element of the social network it represents is also valid. That is, each node $v_i \in V$ is valid for the time period for which the corresponding user u_i or object o_i participates in the social network represented by the graph. Similarly, each edge $(v_i, v_j) \in E$ is valid for the time period that the corresponding entities are connected in the social network.

To incorporate times into the social graph, each element e_i in the graph is annotated with a label that determines the time interval for which the element is valid. In particular, for each element $e_i \in G$, its label is defined as $l(e_i) = (t_{start}, t_{end})$, which implies that element e_i is valid for the time interval $[t_{start}, t_{end})$.

Next, we consider in detail what is determined by the label of each type of element in a social graph. When element e_i with label $l(e_i)$ corresponds to a user u_i, t_{start} is the time point user u_i joined the social network. If the user no longer participates in the network, t_{end} corresponds to the time point the user left the network. Otherwise, if the user is still a member of the social network, $t_{end} = \infty$.

For an object o_i, t_{start} refers to the time point that the object is created and appears in the network, while t_{end} refers to either the time the object stops being valid (for example, if an application is dropped by its creator or an event is removed by its organizer), or ∞ if the object does not have a predetermined expiration time. Note that for some types of objects, for instance events, each object could also be associated with starting and ending time points as part of its description, as the event in Figure 2, where the attributes *start time* and *end time* along with the attribute *date* define the time and duration of the event. For such objects, we consider that their valid time coincides with the time in their description and use the corresponding start and end time as the t_{start} and t_{end} in their label, respectively. For objects that do not include time in their description, such as pages, their label defines the time period that the object appears in the network.

Similarly, for edges, t_{start} corresponds to the time the connection represented by the edge is established, while t_{end} is initially set to ∞ and updated if the connection is dropped with the time it was dropped.

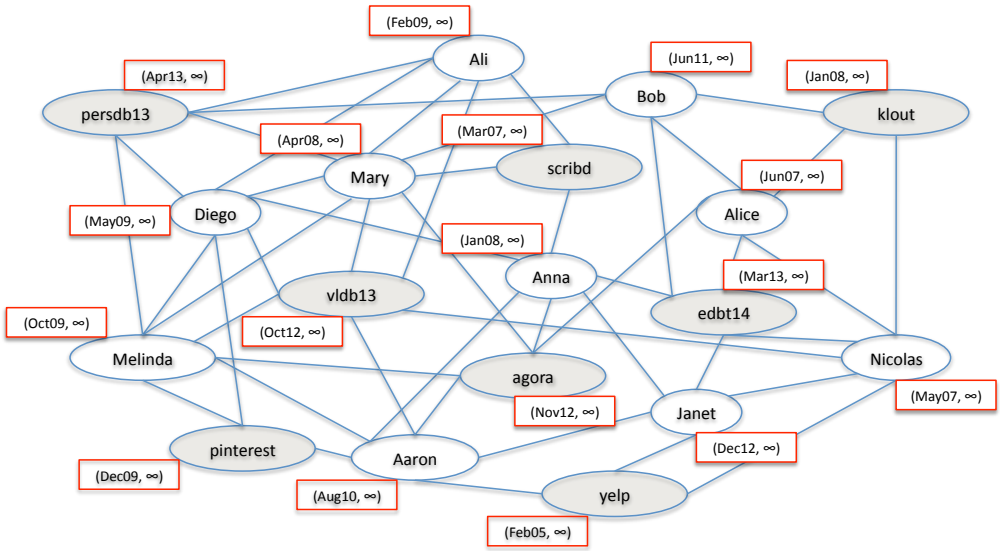

Figure 3: An instance of a social graph in our model depicting entities along with their temporal labels, and the relationships between the entities. For clarity, we skip the labels of the edges.

Figure 3 illustrates our social graph model that incorporates temporal information.

2.3 Query Model

The goal of our system is to support queries for the graph structure that also exploit the time dimension of the elements in the graph. We discern between queries from two different perspectives: *user-centric queries* and *system-centric queries*.

2.3.1 User-centric Queries

Let us first focus on the user perspective and, in particular, on user-centric queries. In such queries, a user u_i is interested in retrieving information about other users or objects that satisfy specific predicates and are connected to the user directly or through their friends. We consider two general categories of user-centric queries. This first category gives priority to the company of the user, while the second one to the objects to be consumed. This way, (i) a user u_i requires all his/her friends that are connected to particular objects, e.g., "*retrieve all my friends that attend all events in Trento with topic Databases*" and (ii) a user u_i requires to retrieve the objects that are connected with a particular set of his/her friends, e.g., "*retrieve all the events that my friends Bob and Mary attend*".

More formally, we define the two categories of user-centric queries as follows:

DEFINITION 1 (USER-CENTRIC QUERY). *Given the graph $G = (V, E)$, $V = U \cup O$, of a social network, a user $u_i \in U$ that corresponds to a node $v_i \in V$ and a set of predicates $P = \{(att_1\ op\ val_1), \ldots, (att_k\ op\ val_k)\}$, a user-centric query $Q(u_i, P)$ is defined as a query that retrieves a set of nodes $V' \subseteq V$ that corresponds:*

(i) [For user-centric queries for friends], *to a set of users $U' \subseteq U$, such that, $u_j \in U'$ iff $((u_i, u_j) \in E)$ and $(\forall o_l$ that satisfies the predicates in P, $\exists (u_j, o_l) \in E)$, and*

(ii) [For user-centric queries for objects], *to a set of objects $O' \subseteq O$, such that, $o_j \in O'$ iff $\forall u_l$ that satisfies the predicates in P and $(u_i, u_l) \in E$, $\exists (o_j, u_l) \in E$.*

Next, we enhance our query model with time by including separate conditions for the validity in specific time intervals for all, or a set of, the elements that are included in a query. For instance, our first query example is modified as: "*retrieve all my friends that will attend all events in Trento with topic Databases during August 2013*".

Let us try to interpret the use of time in this query. The query asks for friends that are associated with events taking place in Trento in August 2013. Thus, for an event e_i to be considered for the query, besides satisfying the predicates "$location = Trento$" and "$topic = Databases$", its valid time determined by its label as the interval $[l(e_i).t_{start}, l(e_i).t_{end})$ should be included within the time period determined by the constraint in the query. That is, the valid time of the event for our example should be in August 2013. Or in the general case, given a time constraint determined by a period $T = [s, d)$ in a query Q, we say that an element e_i (the corresponding user u_i or object o_i) is valid for T, if and only if, $l(e_i).t_{start} \geq s$ and $l(e_i).t_{end} < d$. Note that a reverse interpretation is also possible. That is, one could require the valid time of element e_i to include the time period T determined in the constraint and not the other way. In that case, an element e_i is valid for T, if and only if, $l(e_i).t_{start} \leq s$ and $l(e_i).t_{end} > d$. In the rest of this paper, we always assume the first interpretation for the constraints.

In the query example we use, the time constraint refers to the objects that connect the friends (user nodes) that the query retrieves. One could also use the time to impose constraints on the valid time of the actual user nodes the query retrieves. For instance, "*retrieve all my friends that were valid from 2009 to 2010 and are connected to all events with name = PersDB*" retrieves all my friends that were valid in the time period 2009 to 2010 and have some connection to all events that satisfy the given predicate, even if the events themselves are valid at another time period, i.e., one such event could be PersDB in 2008, and so on.

Similarly, we can interpret the second query example that is modified as: "*retrieve all the events that my friends Bob and Mary will attend in August 2013*". In this case, the constraint is on the events (objects) retrieved, but again we could use constraints on the friends as well.

If the time constraint does not refer to the nodes that constitute the result of a user-centric query, then it needs to be satisfied by the nodes that also satisfy the predicates in P. Therefore, it can be viewed as another predicate to be satisfied. Note that the difference is that the time constraint does not refer to an attribute in the description of the node, but rather to its label. In our first example, for instance, the events o_i that connect the retrieved users should, besides having "*location = Trento*" and "*topic = Databases*", also satisfy "$l(o_i).t_{start} \geq$ August 1 and September 1 $> l(o_i).t_{end}$".

Thus, we focus on queries that apply the time constraint on the nodes in their result and formally define time enhanced user-centric queries. Let us denote the result of a query Q, i.e., the set $V' \subseteq V$, as $res(Q)$.

DEFINITION 2. (TIME-DEPENDENT USER-CENTRIC QUERY). *Given a user-centric query Q and a time interval $T = [s, d)$, a time-dependent user-centric query (Q, T) retrieves all nodes $v_j \in res(Q)$ that are valid for T, i.e., $l(v_j).t_{start} \geq s$ and $l(v_j).t_{end} < d$.*

The definition of time-dependent user-centric queries defines a temporal constraint that checks whether the nodes in the result of Q are valid for a time period T. Note that besides range-based constraints, our model can also support constraints that check whether an object is valid in a specific time point t. In particular, if $s = d = t$, then the constraint that needs to be satisfied is transformed as $l(v_j).t_{start} \leq t < l(v_j).t_{end}$.

2.3.2 System-centric Queries

From the system perspective, system-centric queries target on retrieving information about users that plan to consume objects that satisfy specific predicates or, alternatively, objects that will be consumed by users that satisfy a set of given predicates. Motivated by online shopping applications (e.g., [9]) and targeted advertising (e.g., [1]), system-centric queries identify either sets of users that share some common interests and, for instance, may be interested in a particular product or event the system wants to promote, or similarly, sets of objects that may be of interest to some particular users so that they can choose to promote these objects to them. Based on this motivation, we consider two general categories of system-centric queries. More specifically, (i) the system requires locating the users that are connected to particular objects, e.g., "*retrieve all users that attend all events in Trento with topic Databases*" and (ii) the system requires locating the objects that are connected with particular users, e.g., "*retrieve all the events that users Aaron, Melinda and Diego will attend or have already attended*".

Formally, we define the two categories of system-centric queries as follows:

DEFINITION 3 (SYSTEM-CENTRIC QUERY). *Given the graph $G = (V, E)$, $V = U \cup O$, of a social network and a set of predicates $P = \{(att_1\ op\ val_1), \ldots, (att_k\ op\ val_k)\}$, a system-centric query $Q(G, P)$ is defined as a query that retrieves a set of nodes $V' \subseteq V$ that corresponds:*

(i) [For system-centric queries for users]*, to a set of users U', such that, $u_i \in U'$, iff, for all objects $o_j \in O$ that satisfy the predicates in P, $\exists(u_i, o_j) \in E$, or*

(ii) [For system-centric queries for objects] *to a set of objects U', such that, $o_i \in O'$, iff, for all users $u_j \in U$ that satisfy the predicates in P, $\exists(u_j, o_i) \in E$.*

Similarly to the user-centric case, we augment system-centric queries with time, aiming at retrieving only valid information. The time constraint can be again applied either on the result of the query, or can be viewed as an additional special predicate that concerns the label rather than the description of the node. This way, the enhanced version of the first query example can be formulated as: "*retrieve all the users of the system that are valid in August 2013 and attend events in Trento with topic Databases*". This query declares the interest of the system in retrieving the users that have drawn some attention in participating in events taking place in Trento with subject Databases and were using the system in August 2008. Alternatively, the temporal constraint can refer to the event itself.

In this paper, we do not make any assumptions for the relationships between the retrieved users. One could also envision models where users are connected in the social graph, or are connected via a small number of other users in the graph in order to ensure a strong friendship.

In a similar manner, the second query example can be written as: "*retrieve all the events appearing in the system that the users Aaron, Melinda and Diego will attend in August 2013*", where an event to be considered for the query should be attended by the users at a specific point in time.

In the following, we define the enhanced system-centric queries. Similarly to enhanced user-centric queries, the temporal condition refers to the nodes in the result set of the query.

DEFINITION 4. (TIME-DEPENDENT SYSTEM-CENTRIC QUERY). *Given a system-centric query Q enhanced with a time interval T, a time-dependent system-centric query (Q, T), retrieves all nodes $v_i \in res(Q)$ that are valid for T.*

3. QUERYING THE SOCIAL NETWORK

In this section, we first proceed in describing how our user-centric and system-centric queries are processed against the graph model, and then introduce a simple method for ranking the derived results.

3.1 Query Processing

Let us first consider the user-centric queries for friends. Given the graph $G = (V, E)$, $V = U \cup O$, of a social network, for a query $Q(u_i, P)$, processing proceeds in the following steps:

STEP 1: Retrieve the set of user nodes $u_j \in U$, say U', such that, $\exists(u_i, u_j) \in E$.

STEP 2: Retrieve the set of object nodes $o_l \in O$, say O', such that, o_l satisfies all predicates in P.

STEP 3: From U', remove all nodes u_j, such that, for at least one node $o_l \in O'$, $\nexists(u_j, o_l) \in E$.

STEP 4: The remaining nodes form $res(Q)$.

Similarly, we can enumerate the steps for processing a user-centric query for objects as a sequence of filtering steps.

STEP 1: Retrieve the set of user nodes $u_l \in U$, say U', such that, $\exists(u_i, u_l) \in E$.

STEP 2: From U', remove all nodes that do not satisfy all predicates in P.

STEP 3: Retrieve the set of object nodes $o_j \in O$, say O', such that, $\forall u_l$ retrieved from STEP 2, $\exists(u_l, o_j) \in E$.

STEP 4: The nodes in O' form $res(Q)$.

For time-dependent user-centric queries, an additional step is introduced in which the nodes are filtered according to their labels and their valid times. In particular, for both types of user-centric queries, the additional steps for a time-dependent query (Q, T) are:

STEP 5: From the nodes in $res(Q)$ remove all nodes v_j, such that, $l(v_j).t_{start} < s$ or $l(v_j).t_{end} \geq d$.

STEP 6: The remaining nodes form $res(Q, T)$.

Note that when time does not refer to the retrieved nodes but rather on the nodes against which the predicates in P are checked, in the same step in which the predicates are applied, we can apply the additional time constraint against the labels of the nodes to filter out the non-valid ones.

Let us now consider the processing procedure of a system-centric query $Q = (G, P)$. If Q is a system-centric query for users, the steps are described as follows:

STEP 1: Retrieve the set of object nodes $o_j \in O$, say O', such that, o_j satisfies all predicates in P.

STEP 2: Retrieve the set of user nodes $u_i \in U$, say U', such that, $\exists(u_i, o_j) \in E$, $\forall o_j \in O'$.

STEP 3: The nodes in U' form $res(Q)$.

For a system-centric query for objects, the steps are:

STEP 1: Retrieve the set of user nodes $u_j \in U$, say U', such that, u_j satisfies all predicates in P.

STEP 2: Retrieve the set of object nodes $o_i \in O$, say O', such that, $\exists(u_j, o_i) \in E$, $\forall u_j \in U'$.

STEP 3: The nodes in O' form $res(Q)$.

To deal with time-dependent system-centric queries, two additional steps for filtering are applied. The formal description of such queries is skipped, since it is similar to the description of the time-dependent user-centric queries.

3.2 Ranking

The results returned by our queries so far are defined as a set of nodes with no particular order among them. However, if the cardinality of this set is large, providing a ranking of the results can be very useful from both the user's and the system's perspective. For instance, consider the following query: "*retrieve all my friends that are connected with pages referring to a database topic that are valid in 2013*". The result set includes all friends that have connections to pages that are valid in 2013 with topic "Databases" without any distinction among these friends. However, one can assume that users that connected with a page in 2013 are most likely more actively interested in this page rather than say friends that have connected to the same page in 2009 and are no longer that interested in the same topic. In these scenarios, it might be useful to provide a ranking of results (users, in our example) according to the freshness of the connections between users and objects (pages, here).

Let us again consider the previous example. In this case, even if there is a temporal constraint involved, it does not suffice to provide the ranking we desire. For ranking, we are not interested in the validity of the pages themselves (although this might also be worth considering), but rather on the freshness of the connection between the user and the object we describe in our query through the predicates P. This information is recorded in our graph model on the labels of the edges between the various nodes in the graph. Thus, in our particular example, one can sort the returned results according to the t_{start} of the label of the edges (u_j, o_l) that connect the retrieved users with the qualifying objects based on predicate P. Results related with edges with higher t_{start} values are promoted and placed in higher positions in the ranking, compared to results with edges with lower t_{start} values. Similar examples can be considered for all types of both user- and system-centric queries.

Next, we will present in detail how ranking is applied. In general, our motivation is based on the fact that recently added edges better reflect the current trends and thus, they could contribute more in the ranking of results. Or, in other words, the fresher the connections, the more important the results, and so, the higher their position in the ranking.

We start our description with the case of user-centric queries for friends. Specifically, assume a query $Q(u_i, P)$. The result $res(Q)$ of Q is the set of friends of u_i, $\{u_1, \ldots, u_m\}$, that are connected with the objects O' that satisfy the predicates in P. $ranked\_res(Q)$ of Q is a ranked list of the users in $res(Q)$; ranking is achieved with respect to the labels of the edges that connect the users in $\{u_1, \ldots, u_m\}$ with the objects O' and, in particular, with respect to the time the connections were established. This way, a user u_x precedes a user u_y in the ranking if he/she is the owner of the most recent connection among all connections between u_x and u_y, and O', that is, if he/she is the owner of the edge with the label with the highest t_{start} value among the values in the labels of edges connecting u_x and u_y with O'.

Following the same key points, for a user-centric query $Q(u_i, P)$ for objects, we rank the objects in $res(Q)$ according to the t_{start} values of the labels of the edges that connect the objects with the friends of u_i that satisfy the predicates of P.

DEFINITION 5. (RANKED RESULTS OF USER-CENTRIC QUERIES). *Assume the graph $G = (V, E)$, $V = U \cup O$, of a social network. Given the result $res(Q)$, $res(Q) = \{v_1, \ldots, v_m\}$, $res(Q) \subseteq V$, of a user-centric query $Q(u_i, P)$, $ranked\_res(Q)$ is a ranking of the nodes in $res(Q)$, such that:*

(i) [For user-centric queries for friends], *u_x precedes u_y in $ranked\_res(Q)$, if and only if, there exists an edge between u_x and an object o_z from the objects O' that satisfy P with $l(u_x, o_z).t_{start} > l(u_y, o_{z'}).t_{start}$, $\forall o_{z'} \in O'$, where u_x, u_y correspond to nodes v_x, v_y in $res(Q)$, and*

(i) [For user-centric queries for objects], *o_x precedes o_y in $ranked\_res(Q)$, if and only if, there exists an edge between o_x and a user u_z from the users U' that satisfy P and are connected with u_i, with $l(o_x, u_z).t_{start} > l(o_y, u_{z'}).t_{start}$, $\forall u_{z'} \in U'$, where o_x, o_y correspond to nodes v_x, v_y in $res(Q)$.*

The ranked results of time-dependent user-centric queries are defined in the same way. More specifically, as for the pure user-centric queries, the results consist of either users,

for the queries for friends, or objects, for the queries for objects. Even if the resulting set of returned elements is different, because here we take into account only valid users and objects, the method for determining which elements precede the others follows the same principles.

This approach for ranking users or objects in a social network according to the freshness of their connections with other elements in the network, is used as well for the case of system-centric queries. However, from the perspective of a company that takes advantage of the functionalities of such a system, the procedure of ranking the query results may also exploit other characteristics. For example, one could also envision models where users, or the system itself, prioritize the objects that are available for consumption. In this position paper, we keep this basic model for ranking results, and leave for future work the study of more complex models.

A brute-force method to produce a ranked set of results for either a user-centric or a system-centric query that is dependent on time or not, is the following. First, assign to each element in the query result a score equal to the maximum t_{start} value among the values that appear in the labels of its connections to the nodes that satisfy the predicates of the query. Then, sort the result elements according to that value to construct the ranking of the results. Determining the $ranked\_res(Q)$ for a query Q using such a straightforward algorithm is computationally costly. The focus of our current work is on introducing a top-k variation of the problem along with an algorithm for computing the ranked results in a single phase, i.e., without distinguishing between identifying and ranking results.

4. DISCUSSION AND FUTURE WORK

In this position paper, we focus on the problem of ranking results of queries for the graph structure representing a social network by exploiting the time dimension of the elements, users, objects or connections, in the graph. We distinguish between user-centric and system-centric queries aiming at satisfying the user's and the system's, or company's, information needs, respectively.

In particular, we outline a graph and a query model for the problem and sketch a method for its solution. Clearly, there are many directions for future work including modeling issues, efficient algorithms for computations and implementations in specific contexts. Next, we elaborate on these issues a bit further.

We describe users and objects as sets of predicates of the form (*attribute* = *value*). Other models are feasible as well. For example, going beyond the equality operator, more expressive descriptions, such as range predicates, can be employed. One can also incorporate additional types of objects, like *public posts*, *places* and *groups*, and discern between different types of connection edges between users and objects expressing *likes*, *shares* and *comments*. To capture the dynamic nature of users, objects and their relationships in social networks, labels may consist of a set of mutually exclusive time intervals representing removals and rejoins to the network.

In many cases, the result of a query may contain very few nodes (users or objects) or even may not exist. In such cases, a relaxation approach should be sought for. For instance, instead of requiring a result node to be connected with all the nodes (objects or users, depending on the query) that satisfy a set of given predicates, it may be wise to relax the conditions in order to include in the query result a node that has at least one connection to a node that satisfies the query predicates. Other approaches, such as relaxing the query predicates or approximating them, are also reasonable. Concerning the temporal validity of users, objects and their connections, here we apply hard constraints. That is, we require that a node or an edge that is labeled with a time period T is valid with respect to a given time period T', if T is included in T'. A relaxed version of this approach allows for soft constraints in which T and T' intersect.

Finally, depending on these relaxation schemes, efficient query processing and ranking algorithms need to be designed. For ranking in particular, one may also study more sophisticated multi-criteria ranking schemes, by allowing users to define their own preferences or taking into account the popularity of the returned elements, in terms of number of connections, among the users of the social network. Also, one could classify the connections based on their valid time, as in [5], to form groups of nodes with different temporal characteristics that can be exploited for ranking.

Acknowledgments

The work of the first author is partially supported by the Operational Program "Education and Lifelong Learning" of NSRF-Research Funding Program: Thales (Cloud9), cofinanced by the ESF and Greek national funds. The work of the second author is supported by the project "IdeaGarden" funded by the Seventh Framework Programme under grand n^o 318552.

5. REFERENCES

[1] A. Farahat and M. C. Bailey. How effective is targeted advertising? In *WWW*, pages 111–120, 2012.

[2] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman. Introducing time into RDF. *IEEE Trans. Knowl. Data Eng.*, pages 207–218, 2007.

[3] W. Huo and V. J. Tsotras. Temporal top-k search in social tagging sites using multiple social networks. In *DASFAA (1)*, pages 498–504, 2010.

[4] H. Joho, A. Jatowt, and R. Blanco. A survey of temporal web search experience. In *WWW (Companion Volume)*, pages 1101–1108, 2013.

[5] A. Khodaei and O. Alonso. Temporally-aware signals for social search. In *TAIA*, 2012.

[6] U. Khurana and A. Deshpande. Efficient snapshot retrieval over historical graph data. In *ICDE*, 2013.

[7] G. Koloniari, D. Souravlias, and E. Pitoura. On graph deltas for historical queries. In *WOSS*, 2012.

[8] C. Ren, E. Lo, B. Kao, X. Zhu, and R. Cheng. On querying historical evolving graph sequences. *PVLDB*, pages 726–737, 2011.

[9] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Constructing and exploring composite items. In *SIGMOD*, pages 843–854, 2010.

[10] B. Salzberg and V. J. Tsotras. Comparison of access methods for time-evolving data. *ACM Comput. Surv.*, pages 158–221, 1999.