A Social Network Database that Learns How to Answer Queries[*]

Sara Cohen
Dept. of Computer Science and Engineering
Hebrew University of Jerusalem
sara@cs.huji.ac.il

Lior Ebel
Dept. of Computer Science and Engineering
Hebrew University of Jerusalem
lior.ebel@mail.huji.ac.il

Benny Kimelfeld
IBM Research-Almaden
San Jose, CA 95120
kimelfeld@us.ibm.com

ABSTRACT

Social networks are ubiquitous, with online networks garnering a large portion of Web traffic. Both online and offline, social networks structures are an interesting data source whose importance has been recognized for over a hundred years. Research on social network analysis has dealt with properties of entire networks, in addition to properties of nodes or sets of nodes.

A user queries a social network in pursuit of a desired outcome, such as an expert on a specific medical condition, a set of influential people to promote a new product, or a well-balanced group of database experts to form a program committee. The user may know what the desired outcome is, and may even be able to express it in a formal query language, given the right abstract predicates to represent typical social-network measures (e.g., the importance of a node or its relevance to some keywords). However, choosing the best implementations for these predicates, as well as optimal ranking functions for the results, will often be beyond the abilities of a standard user. In fact, even an expert may experience difficulty with such a task, as the quality of solutions may depend on the precise query at hand, the user preferences, and the nature of the network.

This paper suggests a novel vision of a social network database system. This system incorporates abstract predicates relevant to social networks as primitive building blocks in the query language, and uses machine learning, as an integral part of the query processor, to select and improve upon the predicate implementations. The paper discusses the main features of such a system, as well as the implementation challenges.

1. INTRODUCTION

Online social networks are becoming extremely pervasive, serving as a common method of communication and collaboration. Although such networks are a relatively recent development, offline social networks (i.e., real-life social networks) have been recognized as an important data source for over one hundred years. There has been extensive research on analyzing social networks, stemming from diverse fields including psychology, anthropology, mathematics, and, more recently, computer science.

As social networks are huge, interesting and diverse datasets, the problem of querying social networks has received much attention both in the industry and in academia. Facebook has introduced FQL (Facebook Query Language) which provides an SQL-like language for querying network information. Twitter has advanced search capabilities, allowing users to search based on words, specific people, places, and additional tweet properties. LinkedIn provides job and people search functionalities. Each of these query or search options is rather limited, and provides a solution only for a single social network.

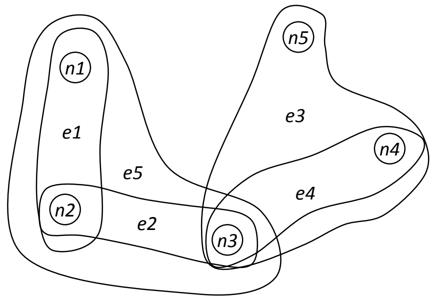
Many software tools are available for network analysis of arbitrary social networks. In fact, Wikipedia [13] lists close to 80 social-network analysis tools. Roughly speaking, these tools usually focus on one or more of the following aspects: visualization, analysis, mining, and database querying and/or manipulation. From a database perspective, social networks can be viewed simply as graphs. Graph databases have been studied extensively [2]. Many query languages for graph databases have been proposed [14], most of which are based on common building blocks such as regular expressions and graph patterns. Queries are generally interpreted in a precise manner (requiring complete satisfaction); however, there has also been work allowing for somewhat looser interpretations of the queries [5, 6].

Databases specifically designed for social networks have also been developed. SoQL [9], is a social-network query language that is based on SQL. SociQL [11], another social-network query language, is also SQL-based, and contains some built-in functions for computing social-network centrality measures. SNQL [8] is based on GraphLog and includes both querying and data manipulation features. SocialScope [1] allows ranking of query results based on social relations. Finally, Dries et al. [3] combine social-network querying, data mining and clustering capabilities.

With so many languages and systems already developed, it is perhaps surprising that this paper introduces yet another vision of a social-network database system (and even dares to label such an idea "outrageous"). Actually, when looking over current work, it seems that there is quite a chasm between the types of queries easily expressible in graph (or social-network) databases, and natural queries that arise

[*]Sara Cohen and Lior Ebel were partially supported by the Israel Science Foundation (Grant 143/09) and by the Ministry of Science and Technology (Grant 3-8710).

6th Biennial Conference on Innovative Data Systems Research (CIDR '13) January 6-9, 2013, Asilomar, California, USA..



Nodes

n1	Name = Alice
	Age = 32

n2	Name = Bob
	High-school = LVHigh

n3	Name = Cindy

n4	Name = Dan
	Age = 35

n5	Name = Erica
	Hobby = Bungee jumping

Edges

e1	({n1},{n2})	Label = follows
e2	{n2,n3}	Label = friends

e3	{n3,n4,n5}	Label = co-authors
		Title = Bungee for Fun
		ISBN = 12345

e4	{n3,n4}	Label = friends
e5	({n3},{n1,n2})	Label = email Content = Check out my new book…

Figure 1: A tiny portion of a social network

from the field of social-network analysis.

Current proposals focus mostly on precise, database style queries (e.g., find all common co-authors of two specific authors, find people 2-hops away from a specific node, find pairs of nodes whose connecting path matches a specific regular expression), without the capability to express richer imprecise queries that are more in the spirit of social-network analysis.[1]

On the other hand, typical social-network questions include expert search, friend recommendation and community recognition. Each of these problems, along with many others, has been studied individually (e.g., [7,15]) and quite extensively. Often, there are specific network properties that seem to contribute positively towards preferable solutions to these problems. For example, expert recommendation may take into consideration centrality of target nodes, and friend recommendation may take into consideration the closeness of the source and target nodes (to leverage common social-network properties, such as triadic closure and homophily). However, the optimal interpretation of such properties is often unknown, as it is not obvious how one should best measure centrality or closeness. Therefore, for each new social-network problem, many different instantiations of ranking functions for network properties are considered, often together with a machine-learning mechanism, to determine the parameters of these functions as well as the effective amalgamation of their outputs.

The database system envisioned in this paper will bridge the gap by bringing together (1) a declarative query language, (2) abstract social-network ranking functions, along with efficient instantiations and (3) a learning mechanism, that allows the best instantiations to be learned, per query (from user feedback), if so desired. Such a system will accommodate a variety of social-network-oriented questions. It will also be extensible, to allow for new implementations of social-network measures to be seamlessly integrated into the system, allowing all types of queries to take advantage, immediately, of such advances.

In the remainder of this paper, we introduce the main underlying ideas of our system, by focusing on the querying mechanism. We also briefly discuss some of the many challenges that will arise in actually implementing such a social database system, which require rethinking many core database issues.

2. QUERY LANGUAGE

Data Model. A social network is simply a graph. The nodes represent people, and can have a variety of attributes, associated with attribute values. Edges are labeled, and may be directed or undirected, and weighted or unweighted. In addition, to easily represent groups of people and interactions, edges may be (directed or undirected) hyper-edges. Edge weights can be user defined, or system generated (e.g., based on frequency of communication). As edges represent various types of relationships and interactions among people they can also be associated with attributes and values.

To demonstrate, Figure 1 depicts a tiny portion of a social network $\mathcal{N}$. Nodes $n_1, \ldots, n_5$ are associated with attributes and values. For example, n_1 is associated with the attribute-value pairs "Name = Alice" and "Age = 32". In our system, we also assume that each node has an identifier (which may be system generated), assigned to the attribute named "id." The network $\mathcal{N}$ also contains five edges. Observe that all edges are labeled, and some edges are associated with additional attribute-value pairs. Edges also have identifiers (again, associated with an "id" attribute). There are several different types of edges in $\mathcal{N}$: e_1 is a directed edge, $e_2, e4$ are undirected edges, e_3 is an undirected hyper-edge and e_5 is a directed hyper-edge.[2]

Queries. Due to space limitations, we do not provide a complete description of syntax or semantics. Instead, we explain the main features of our language, and present several illustrating examples.

Our query language uses an SQL-style syntax. Query variables can range over nodes, sets of nodes or edges. Since a social network may contain a variety of types of relationships (i.e., edges), not all of which may be of interest for a particular query, queries have special `PROJECT` clauses that define the sub-network of interest. Within the `WHERE` clause, we use a dot-notation to access attribute values of nodes/edges. For convenience, we use labels of edges as boolean predicates

[1]SociQL [11] allows some limited filtering by specific centrality measures as part of a special FILTER BY clause.

[2]Of course, all edges are simply special cases of directed or undirected hyper-edges.

within our queries, as will be apparent below. Queries may also have `RANK BY` and `LIMIT` clauses.

A unique aspect of the language is the presence of special built-in functions, used as building blocks for expressing interesting queries over the social network:[3]

- `imp`: measures the importance of a node;
- `sim`: measures the similarity of a set of nodes;
- `cls`: measures the closeness of a set of nodes;
- `infl`: measures the influence of a set of nodes on the rest of the network;
- `dscr`: measures the strength of the relationship between a set of nodes and given text.

There are many different ways to implement these functions. For example, `imp` can be implemented using any one of many different centrality measures [4]. As another example, `cls` can be implemented using functions such as Adar/Adamic, rooted PageRank, the Katz measure, or others [7]. In fact, as discussed above, we expect a system to include several different implementations of each built-in function.

Our built-in functions return a numerical value that can be used for ranking. However, we will also use these functions as boolean predicates within a `WHERE` clause. The query processor will interpret these predicates by choosing a threshold for the minimal return value of the function. (Choosing a threshold, as well as choosing an implementation of a function and parameters thereof, can be done manually through user customization or automatically by means of machine learning, as we discuss later on.)

We demonstrate the main features of our language by a series of examples. Each example expresses a standard social-network analysis problem within our simple declarative language. Note that we steer clear of queries that have a classical relational database flavor, as they are not of special interest in this language. However, such queries can be easily expressed in the language.

Example 1 (Link Prediction [7]). *Given a node $n \in \mathcal{N}$, the link prediction problem is to find nodes that n is currently not friends with, who are likely to become friends with n in the near future. The ability to find such nodes is useful for friend recommendation.*

The following query finds the top-10 best results for predicting new links for node 17. Note the use of the `FROM` *clause to state that n and m range over nodes. Note also that* `friends` *is used as a boolean predicate (as mentioned earlier) which returns true if the set of nodes in its argument is an edge labeled* `friends` *in the graph.*

```
SELECT m
FROM NODE n, NODE m
WHERE n.id = 17 and not friends({n,m})
LIMIT 10
RANK BY sim({n,m})
```

Example 2 (Expert Search [15]). *Given some string of text t, the expert search problem is to find a node that is an expert on the topic t. In order to return interesting results, the following query ranks by a combination of the expertise of the node on t, and its general graph importance. The* `WHERE` *clause uses the built-in function* `dscr` *in a boolean fashion to require some minimal expertise in the topic "XML", and again uses* `dscr` *in the* `RANK` *clause as a ranking factor.*

```
SELECT n
FROM NODE n
WHERE dscr({n},"XML")
LIMIT 10
RANK BY {imp({n}), dscr({n},"XML")}
```

Example 3 (Program Committee). *In the final example, we consider the problem of forming a program committee. All members should be experts in the topic, should be people of importance, and should be somewhat diverse. We use* `NODESET` *in the* `FROM` *clause to declare a variable ranging over sets of 15 nodes. Note also the use of* `PROJECT` *to restrict ourselves to the portion of the graph containing edges that indicate authorships.*

```
SELECT N
FROM NODESET(N,15)
PROJECT ON EDGES WHERE label = "co-authors"
WHERE dscr(N,"ACID, views...")
LIMIT 10
RANK BY {imp(N), dscr(N,"ACID, views..."),-sim(N)}
```

In summary, the main unique aspects of this query language are that it combines a declarative style with built-in social-network predicates (which allow a variety of interesting queries to be expressed), while decoupling the implementation of these predicates from the query declarations.

Answering Queries. In principle, query processing should proceed in a manner similar to processing queries in other languages, such as SQL. However, the use of our special built-in functions gives rise to three distinct problems.

- First, there are many different possible useful implementations for each function. Assuming that there are several implementations, which should be chosen for use during the evaluation of a specific query?
- Second, when using a built-in function as a boolean predicate within a `WHERE` clause, what threshold should be chosen to determine satisfaction of the predicate?
- Third, when several functions are specified within the `RANK BY` clause, what should be the precise combination to be used when ranking results?

A minimal requirement from the system is to allow the user to tune her query by specifying the answers to all three questions, that is, choosing function implementations from among those available, setting a threshold, and determining the precise ranking function. Thus, the user can have complete control on the interpretation of her query by the query processor.

We envision a system that goes significantly beyond this minimal requirement. When the user does not know how to choose the best implementation of her query, she can request the system to make default choices for her, and then, to learn from feedback about the answers that the system returns.[4]

[3]Additional functions, measuring other social-network properties, are also possible.

[4]In some circumstances the user feedback may be implicit, as in the case of link-prediction queries.

Using machine learning, the databases system can attempt to improve upon the query results (i.e., improve the three choices above), when the query is rerun. We note that there is an inherent time/effectiveness tradeoff, as learning of each query by the system is quite costly. For queries that will be run often, and for which extremely effective result ranking is important, a learning process is highly useful. Thus, in a nutshell, our vision is of a *social-network database system with a declarative query language including social-network functions, that can also learn how to answer queries.*

We note that the problem of learning queries from query results has received quite a bit of attention lately (e.g., [10]). Such work generally assumes that the user provides some (positively and/or negatively) labeled answers, and the query (in a given query language) is learned. In contrast, in our system the basic structure of the query is known. Instead, the best implementation of the built-in predicates (among those available), as well as their combinations within the ranking functions, are learned.

3. MAIN CHALLENGES

Implementing a database system for social networks that learns how to answer queries is extremely challenging. In fact, such a system requires rethinking many core database issues. We detail some of the main issues.

Efficient Query Evaluation. First, and foremost, enabling efficient query evaluation is a huge challenge. At the very least, this requires the built-in functions to be efficiently computed. However, previously considered methods of measuring centrality of a node, closeness, etc., are often quite costly. For example, there have been several works [12] which have studied the efficient implementation of path distance for pairs of nodes within a social network. Thus, even this simplest implementation of closeness (and even for only two nodes) is already a challenge due to the mammoth sizes of social networks, and their constant evolutions.

Learning Mechanism. Effectively learning from user examples is another challenge. Queries may have several different parameters that must be learned, and users may provide only limited feedback. Choosing an effective learning mechanism, and integrating it efficiently within the query processor, is already a non-trivial task. Additionally, providing guarantees, such as eventual convergence on the optimal solution, is even more challenging. Determining how to solicit user feedback on specifically chosen examples, in order to speed up the learning process, is also an interesting problem in the given setting. Finally, while learning is typically performed for a specific query, an interesting and important problem is to leverage the results of one learning process to speed-up the learning of parameters for a different query.

Privacy. Standard online social networks provide a very limited array of privacy controls for their users. For example, Facebook users can choose the allowed audience for various information (including *public*, *friends*, *only me* or a custom specified list). However, we envision a richer privacy mechanism to be of importance. For example, the user may want to allow herself to be found within specific contexts, or by specific people. As a simple example, a dentist who participates in a social network may desire to be a possible result for queries relating to dentistry (but may wish to remain anonymous for other types of queries). As another example, a database researcher may be willing to be found by any other person who is of importance in the database field, but not by others. Developing a formalism for such privacy controls is a challenge, as is determining how to evaluate the built-in functions without inadvertently exposing private information.

Advanced Features. There are many advanced features that would be invaluable within the context of our social-network database. One such feature is the ability to ask hypothetical questions (e.g., what will be the effect on the network if Bob and Alice are no longer friends). Another is the effective management of time and provenance within the network (e.g., why did we suggest Bob as a friend for Alice). Finally, integrating aggregation into the query language, and allowing efficient computation of various statistics over the network (and updating them as the network evolves), is also an interesting challenge.

4. REFERENCES

[1] S. Amer-Yahia, L. V. S. Lakshmanan, and C. Yu. Socialscope: Enabling information discovery on social content sites. In *CIDR*, 2009.

[2] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1–39, 2008.

[3] A. Dries, S. Nijssen, and L. De Raedt. A query language for analyzing networks. In *CIKM*, 2009.

[4] L. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1979.

[5] G. Grahne and A. Thomo. Regular path queries under approximate semantics. *Ann. Math. Artif. Intell.*, 46(1-2):165–190, 2006.

[6] Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *PODS*, 2001.

[7] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *CIKM*, 2003.

[8] M. S. Martín, C. Gutierrez, and P. T. Wood. Snql: A social networks query and transformation language. In *AMW*, 2011.

[9] R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *ICDE*, 2009.

[10] S. Staworko and P. Wieczorek. Learning twig and path queries. In *ICDT*, pages 140–154, 2012.

[11] D. F. S. Suarez. *SociQL: A Query Language for the Social Web*. PhD thesis, University of Alberta, 2011.

[12] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *CIKM*, 2007.

[13] Wikipedia. Social network analysis software — Wikipedia, the free encyclopedia, 2012. [Online; Accessed July 2012].

[14] P. T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, Apr. 2012.

[15] J. Zhang, J. Tang, and J. Li. Expert finding in a social network. In *Advances in Databases: Concepts, Systems and Applications*, volume 4443. Springer, 2007.