Stream Querying and Reasoning on Social Data

Jayanta Mondal Amol Deshpande

Department of Computer Science, University of Maryland, College Park MD 20742
{jayanta, amol}@cs.umd.edu

1. SYNONYMS

Continuous query processing; Temporal analytics; Dynamic social networks; Incremental computation.

2. INTRODUCTION

Since the inception of online social networks, the amount of social data that is being published on a daily basis has been increasing at an unprecedented rate. Smart, GPS-enabled, always-connected personal devices have taken the data generation to a new level by making it tremendously easy to generate and share social content like *check-in* information, *likes*, *microblogs* (e.g., Twitter), multimedia data, and so on. There is an enormous value in reasoning about such streaming data and deriving meaningful insights from it in real-time. Examples of potential applications include advertising, sentiment analysis, detecting natural disasters, social recommendations, personalized trends, spam detection, to name a few. There is thus an increasing need to build scalable systems to support such applications. Complex nature of social networks and their rapid evolution, coupled with the huge volume of streaming social data and the need for real-time processing, raise many computational challenges that have not been addressed in prior work.

Social network data comprises of two major components. First, there is a network (*linkage*) component that captures the underlying interconnection structure among the entities in the social network. Second, there is *content data* that is typically associated with the nodes and the edges in the social network. The social network data *stream* contains updates to both these components. The structure of the network may itself change rapidly in many cases, especially when things like webpages and user tags (e.g., Twitter *hashtags*) are treated as nodes of the network. However, most of the social network data stream consists of updates to the data associated with the nodes and the edges, e.g., status updates and other content uploaded by the users, communication among the users, and so on. There is interest in performing a wide variety of queries and analytics over such data streams in real time. The queries can range from simple publish-subscribe queries, where a user is interested in being notified when something happens in his friend circle, to complex anomaly detection queries, where the goal is to identify anomalous behavior as early as possible.

In this paper, we present an introduction to this new research area of *stream querying and reasoning* over social data. This area combines aspects from several well-studied research areas, chief among them, social network analysis, graph databases, and data streams. We provide a formal definition of the problem, survey the related prior work, and discuss some of the key research challenges that need to be addressed (and some of the solutions that have been proposed). We note that we use the term *stream reasoning* in this paper to encompass a broad range of tasks including various types of analytics, probabilistic reasoning, statistical inference, and logical reasoning. We contrast our use of this term with the recent work by Valle et al. [104, 105] who define this term more specifically to refer to integration of logical reasoning systems with data streams in the context of the Semantic Web. Given the vast amount of work on this and related topics, it is not our intention to be comprehensive in this brief overview. Rather we aim to cover some of the key ideas and representative work.

3. PROBLEM DEFINITION

An online social network is defined to be a community of people (called *users*) connected via a variety of social relations, that use online technologies to communicate with each other and share information. Social data is defined to be the data arising in the context of a social network that includes both the embedded structural information as well as the data generated by the users. Online social networks continuously generate a huge volume of such social data that includes both structural changes to the network and updates that are associated with the nodes or the edges of the network. The task of "stream querying and reasoning" refers to ingesting and managing such continuously generated data, and querying and/or reasoning over it in real-time as the data arrives.

To make the discussion more concrete and formal, let $\mathcal{G}_t(V_t, E_t)$ denote the underlying social graph at time t, with V_t and E_t denoting the sets of nodes and edges at time t respectively. In general, $\mathcal{G}_t$ is a heterogeneous, multi-relational graph that may contain many different types of nodes and may contain both directed and undirected edges (Figure 2 shows an example graph). Along with nodes representing the users of the network, V_t may include other types of nodes, e.g., nodes representing communities or groups, user tags, webpages, and so on. Similarly, E_t includes not only symmetric *friendship* (or analogous) edges, but may include asymmetric *follows* edges, *membership* edges, and other types of semi-permanent edges that are usually in existence from the time they are formed till the time they are deleted (or till the current time). We distinguish such edges from *transient* edges that can be used to capture specific interaction between two nodes in V_t (e.g., a message being sent from one node to another). A transient edge is typically

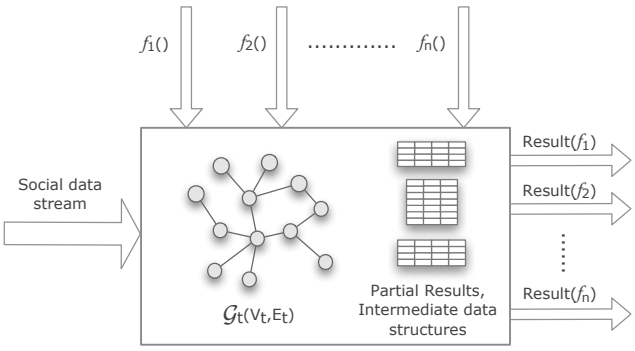

Figure 1: High-level overview of a stream querying system

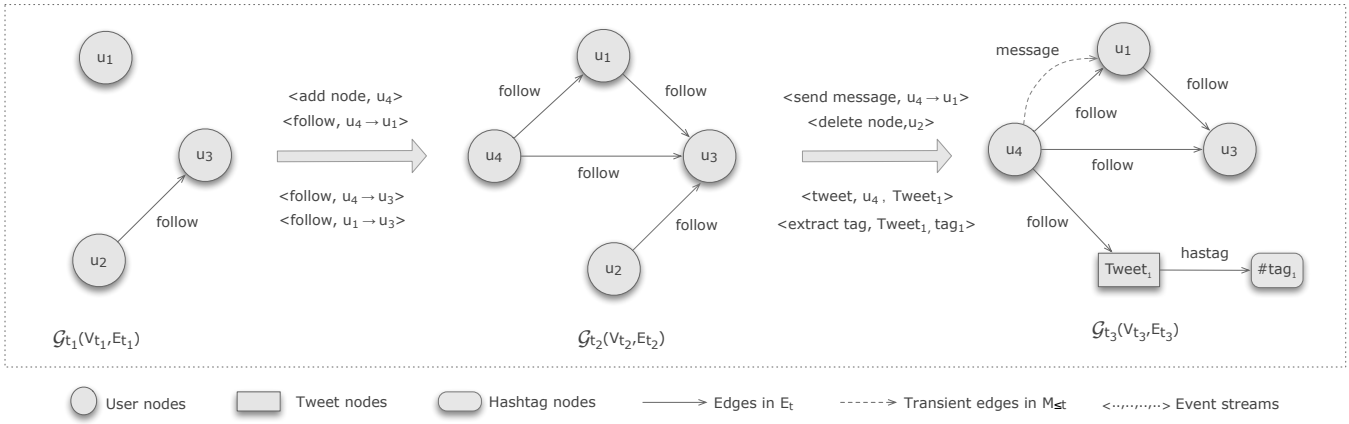

Figure 2: Example of a multi-relational, heterogenous dynamic graph

Notation	Description
$\mathcal{G}_t(V_t, E_t)$	current state of the network
$M_{\leq t}$	transient edges generated till time t
$f_1(), f_2(), ...$	stream querying or reasoning tasks
$\mathcal{N}^k(v)$	k-hop ego network of node v

Table 1: Notation

time-stamped and is only valid for the specific time instance. To allow us to clearly distinguish between these two types of edges, we do not include such transient edges in E_t; instead, we use $M_{\leq t}$ to denote all such transient edges that were generated from the beginning (i.e., from time 0) till time t. This distinction is not necessary, but affords clearer distinctions between different types of stream reasoning tasks in many cases.

The information associated with the nodes and edges can be captured through a set of *key-value* pairs (also called *attribute-value* pairs) associated with them. We once again can make a distinction between semi-permanent information associated with the nodes or the edges (e.g., user *names*, *interests*, or *locations*) and transient information associated with them (e.g., *status updates*). The former type of information can be seen as being valid for a given time period, whereas the latter is typically associated with a single time instance.

Given this, we define a stream reasoning or querying task to be a declaratively-specified query or an analysis or reasoning task that is posed (submitted) once by the user, but is executed continuously (or periodically with a user-specified frequency) as updates arrive into the system (Figure 1). Along with a task, denoted $f()$, the user must specify what forms the *input* to the task, when to compute the *output*, and when to *return* the output to the user.

In many cases, the input is the *current* graph, i.e., the input is $\mathcal{G}_t(V_t, E_t)$ (that is continuously changing). An example of such a task is *dense subgraph maintenance* [18] where the goal is to compute and maintain the dense subgraphs in a dynamically changing graph. In other cases, the input to $f()$ may be defined using a *sliding window*, i.e., it may be defined as the set of all updates that arrived in recent past. An example of such a task is continuously identifying dense subgraphs in the graph formed by all message edges over say the last 24 hours (i.e., the input to the task is $M_{\leq t} - M_{\leq (t-24\ hours)}$). As time progresses, the window slides and new message edges will be added to the graph and old message edges (that fall out of the window) will be deleted.

The second key issue is when to compute the output and when to return it to the user. In some cases, the user may desire continuous execution of the query, i.e., for every relevant change in the input, $f()$ needs to be recomputed (either from scratch or incrementally). Anomaly detection queries typically need to be executed in this fashion since anomalies must be detected as soon as they are formed. But in other cases, the user may specify a frequency with which to execute the query or the task (e.g., every hour or every day). Finally, for simplicity, we will assume that the user should be notified any time the output of $f()$ is computed and is different from the prior output. However, in many cases, the output may need to be returned to the user only when he asks for it. In those cases, partial pre-computation of the query results (with the rest of the processing performed at query time) becomes a possibility.

4. HISTORICAL BACKGROUND

Stream querying and reasoning over social networks combines aspects from several different research areas that have been very well studied over the last few decades. Here we will provide very brief background on three of the most closely related research areas: social network analysis, data streams, and graph databases. A more detailed background, including references to related work, can be found in an extended version of this article [1].

Social network analysis: Social network analysis, sometimes called *network science*, has been a very active area of research over the last decade, with much work on network evolution and information diffusion models, community detection, centrality computation, and so on. We refer the reader to well-known surveys and textbooks on that topic (see, e.g., [93, 97, 31]). There has been an increasing interest in dynamic or temporal network analysis in recent years, fueled by the increasing availability of large volumes of temporally annotated network data and the real-time requirements of various popular online services. Such analysis has the potential to lend much better insights into various phenomena, especially those relating to the temporal or evolutionary aspects of the network. Many works have focused on designing analytical models that capture how a network evolves, with a primary focus on social networks and the Web (see, e.g., [6, 74, 72]). There is also much work on understanding how communities evolve, identifying key individuals, locating hidden groups, identifying changes, and visualizing the temporal evolution, in dynamic networks [28, 101, 100, 57, 84, 22, 76, 44, 96, 32, 17, 10]. Most of that prior work, however, focuses on off-line analysis of static datasets. Berger-Wolf et al. [28, 101], Tang et al. [100] and Greene et al. [57] address the

[1] http://www.cs.umd.edu/~jayanta/papers/SRQ-ESNAM.pdf

problem of community evolution in dynamic networks. McCulloh and Carley [84] present techniques for social change detection. Asur et al. [22] present a framework for characterizing the complex behavioral patterns of individuals and communities over time. In a recent work, Ahn et al. [11] present an exhaustive taxonomy of temporal visualization tasks. Ren et al. [96] analyze evolution of shortest paths between a pair of vertices over a set of snapshots from the history. More generally a network analyst may want to process the historical trace of a network in different, usually unpredictable, ways to gain insights into various phenomena. We note that, although some of the above work focuses on temporal aspects of social network data, none of it tries to do stream query or reasoning in an online fashion. Instead the focus in this work is on offline analysis of static datasets.

Data streams: Data stream management is another research area that has seen tremendous amount of work over the last decade (see [5, 91, 54] for comprehensive surveys), resulting in several data management systems being built (e.g., NiagaraCQ [92], TelegraphCQ [81], STREAM [87], Aurora [3], HiFi [42], to name a few). That work was spurred by the increasing volumes of data being generated in an online fashion that needed to be processed and analyzed in real-time. continuous query evaluation. Unlike a one-time query, for a continuous query (also sometimes called a **standing** query), the system is expected to keep the answer up-to-date as new data arrives. Such queries are often observed in *publish-subscribe systems* or *complex event processing systems* [42, 43, 34, 39, 49, 48, 46, 8, 107], where a centralized system is typically in charge of ingesting the published data from the data sources, and deciding if any updates need to be sent to the subscribers (whose subscriptions can be viewed as continuous queries) or if any events need to be generated. Efficiently supporting continuous queries over rapidly changing data streams has seen much work over the last decade.

Several SQL extensions have also been proposed to express continuous queries over data streams. Similarly, languages have also been designed for specifying event patterns to be matched against data streams (e.g., SASE [61]). Continuous query processing also bears strong resemblance to materialized view maintenance, an area that has also seen much work [59]. The key difference between the two research areas has been that: continuous query processing systems are designed to simultaneously support large numbers of relatively simple queries over highly dynamic data, whereas view maintenance techniques usually focus on a small number (usually just one) of more complex queries. The former also tend to build intermediate data structures like *predicate indexes* to efficiently identify the queries whose results are affected by new updates. Another line of work has focused on development of *one-pass* algorithms that can incrementally compute some quantities of interest over very large volumes of data (e.g., statistics or aggregates) while using very small amounts of memory (see, e.g., [91]).

Graph databases: Since social networks are naturally represented as graphs, specialized graph data management systems are a natural option to store social network data. There has been much work on single-site graph databases (e.g., [16, 60, 62, 99, 64], Neo4j [1]), and in recent years, on distributed graph databases and programming frameworks for specifying batch analysis tasks over graphs (e.g., HyperGraphDB [66], InfiniteGraph [67], GraphBase [56], Trinity [30], Pegasus [71], Pregel [82], GraphLab [79, 78], Giraph [20], etc). There is also much work on executing specific types of queries efficiently over graphs (both in centralized or distributed settings) through strategic traversal of the underlying graph, e.g., reachability [103, 63, 68, 108], keyword search queries [55, 102, 29, 70], subgraph pattern matching [64, 51, 35, 65], shortest paths queries [106, 53, 58], etc. However, distributed management of dynamic graph data is not as well-studied, especially in the data management research community. Recently there has been an increasing interest in large-scale, distributed graph data management, with several new commercial and open-source systems being developed for that purpose. Some of the key systems include Neo4j [1], HyperGraphDB [66], InfiniteGraph [67], GraphBase [56], Trinity [2], Pegasus [71], Pregel [82], GraphLab [79, 78], Giraph [20], etc. There have also been several programming frameworks that have been proposed There has also been much work on executing specific types of queries or performing specific types of analysis, e.g., subgraph pattern matching queries [35, 65], reachability queries [103, 63, 68], data mining [71], modeling network evolution [6, 74, 72], and so on. But those works either have limited focus or, in the case of Pegasus [71], are meant for batch processing.

5. PROPOSED SOLUTION AND METHODOLOGY

The area of stream querying and reasoning over social networks is still in its infancy, and as a result, the research in this area is somewhat fragmented with several ongoing attempts at unifying the different research themes. Here we begin with a broad classification of the different types of stream querying and reasoning tasks and give examples of different types of tasks that have been studied in prior literature. We then discuss some of the key research challenges in effective stream querying and reasoning that need to be addressed.

5.1 Classifying Tasks by Scope

Here we attempt to classify stream reasoning and querying tasks by their input scope, i.e., what data forms the input to the task at any time. Broadly speaking, there are two crucial dimensions along which the tasks may differ.

5.1.1 Temporal Scope

The first key dimension captures the temporal scope of the task, and has a direct impact on the amount of state that must be stored, and reasoned about.

Entire stream: At one extreme, the temporal scope of a stream reasoning task may stretch from the beginning of the stream to the current time. Note that, not all the data generated so far may be of interest – e.g., the task may only see a subset of the data by choosing to focus only on certain attributes of the nodes or edges. However, the data of interest may have arrived into the system at any point in the past. For example, in a social network with location data, a stream reasoning task may wish to process all the location updates ever produced by a user for predicting future user movements. We expect such types of stream reasoning tasks to be somewhat uncommon given the large volumes of data generated in most online social networks.

Current state of the network: Many stream reasoning tasks will take the current state of the network (i.e., $G_t(V_t, E_t)$) as the input. An example of this task is online dense subgraph maintenance [18] where the goal is to maintain the dense subgraphs of the current social network at all times.

Sliding window: The third alternative, that falls in between the two extremes above, is that the reasoning task defines a sliding window on the data stream, and the input consists of all updates that arrive during that window. For instance, one may be interested in analyzing all messages that were exchanged during the last 24 hours

among the users of a networks to identify anomalous behavior in real-time. Another example of such a task is detection of personalized trends where the goal is to find the most commonly seen words or phrases in the recent status updates or blog posts by the friends of a user.

5.1.2 Network Traversal Scope

The second key dimension is what we call *network traversal scope* of a query, which refers to the portion of the network that provides the input to a stream reasoning query or task.

Global scope: Many stream reasoning tasks require reasoning over the entire network. An example of such a task is computation of *PageRank* (or other centrality measures like *betweenness centrality*, *eigenvector centrality*, etc.). Dense subgraph maintenance task discussed above is also an example of a task with global scope.

Ego-centric scope: On the other hand, in many cases, a reasoning task or a query may only focus on a local neighborhood in the network, often termed *ego networks*. For example, if the goal is to identify *social circles* for a user [83], then only a 1- or 2-hop neighborhood around the user may be of interest (Figure 3). Personalized trend detection task, discussed above, is another example of such a task. Note that, in many cases, we may want to execute the same task for every node in the network (e.g., we may wish to do continuous trend detection for every user of the social network), and in total, updates in the entire network may need to examined. However, those should be treated as separate tasks each of which is ego-centric in scope. The most common example of an ego network is the network over the immediate set of neighbors of a node. However, in general, an ego network of node could be defined as *k-hop neighborhood* containing all nodes reachable within k hops from the node (and all the incident edges among those).

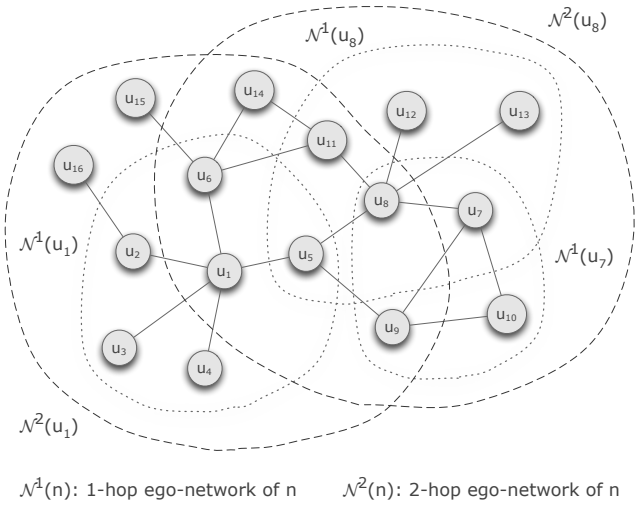

Figure 3: Stream queries often have ego-centric scope: figure shows 1-hop ego-networks of $u_1, u_7,$ and u_8, and 2-hop ego-networks of u_1 and u_8.

5.2 Types of Stream Reasoning Tasks

Next we attempt to provide a categorization of different stream reasoning and querying tasks by type. Given the wide variety in the stream reasoning tasks of interest, unlike the categorization by scope, the categorization that follows is less precise and not fully disjoint. Our intention here is not to be comprehensive, but rather to discuss some representative stream reasoning tasks.

5.2.1 Publish-subscribe Queries

Perhaps the simplest kind of queries over streaming data are what are commonly referred to as *publish-subscribe* queries. These queries form a subclass of the more general class of *event monitoring* queries, where the users specify events or updates of interest and they should be notified as soon as a matching event is detected in the data stream. We make a loose distinction between simple event monitoring queries (what we call publish-subscribe queries), and more complex event monitoring or anomaly detection queries (discussed subsequently). For publish-subscribe queries, the events are typically defined over one or a few data stream updates (i.e., they have very limited temporal and traversal scopes). For example, a user may be interested in tweets that contain a particular key word, or a user may want to know as soon as a friend is online. In a location-enabled social network, a user may be interested in getting notified when one of his friends checks-in in a nearby restaurant or cafe. The key challenge with executing simple event monitoring queries is not so much the complexity of detecting the events, but rather dealing with the very large update rates as well as a very large number of queries.

5.2.2 Complex Event Processing (CEP)

On the other hand, in complex event processing, the events (often called *patterns*) to be detected often have larger temporal or network traversal scopes, or both. For example, actions done on the same object by two users in a social network might be temporally correlated in applications involving influence computation. For example, detection of trending news in a social network might require traversal beyond users' immediate friends, i.e., friends of friends' and so on. Hence, unlike simpler publish-subscribe queries, efficiently detecting the events can be a major challenge in CEP. An example of such a query is a continuous subgraph pattern matching query, where the goal is to detect matches to a given query graph in real-time. Choudhury et al. [41] use such queries for continuous detection of accidents from incoming traffic information. Complex event processing systems often support specification of the events using a high-level declarative language. For example, in recent work, Anicic et al. [19] proposed a language called EP-SPARQL that extends the SPARQL query language with support for specifying complex event processing queries over RDF data streams. Similarly, Mozafari et al. [90] present a language for detecting hierarchical patterns over hierarchical data (e.g., XML data), that may be generalizable to graph-structured data as well. Choudhury et al. [41] have discussed continuous detection of accidents from incoming traffic information such application in their work on continuous queries for multi-relational graphs. They present the problem as a subgraph pattern matching problem on streams in real-time. Zhao et al. [50] have also looked at a similar problem where they use an incremental algorithm to compute the changes to the existing pattern match rather than expensive re-computation. Gao et al. in their work on LBSN [52]define two behaviour models: (i) a historical model (HM) and (ii) a social-historical model (SHM) and compare their performance in order to provide meaningful location related service. Chandramouli et al. [38] has looked into problem of streaming recommendation problem that supports real-time incremental processing to support on-demand recommendation using stream processing system.

5.2.3 Anomaly Detection

Anomaly detection queries can be seen as a form of complex event processing, however, due to their importance, we discuss them separately. The goal with real-time anomaly detection is to identify anomalous behavior in a dynamic network as quickly as

possible. Two issues need to be addressed: (1) how to define what constitutes an "anomaly"? (2) how to efficiently detect anomalies in presence of very high data rates? Generally speaking, anomalous behavior can be defined as behavior that deviates significantly from *normal* behavior. However, in highly dynamic and rapidly changing environments like an online social network, there is often no clear definition of normal behavior, making it a challenge to identify anomalous behavior. There have been many proposals for defining anomalous behavior in social networks over the years. For example, Akoglu et al. [14] present an approach called Oddball, that is based on analyzing the ego-networks of the nodes in the network. Aggarwal et al. [7] propose a probabilistic algorithm that maintains summary structure models about graph streams to detect outliers. We refer the reader to the tutorial by Akoglu and Faloutsos [13] for a more comprehensive discussion of different anomaly detection algorithms.

Perhaps because of a lack of a clear definition of an anomaly, there is much less work on efficient techniques for real-time anomaly detection. From the efficiency perspective, an important issue is the scope (both temporal and network traversal) of an anomaly detection task. For example, if the goal is to identify users with anomalous behavior, then the network traversal scope could be limited to ego networks of the users. However, in many cases, detecting anomalous behavior may require global reasoning over the entire network.

5.2.4 Continuous Aggregates/Statistics Computation

In these types of queries, the goal is to incrementally maintain or compute an aggregate or a statistic over the network [86]. An example of such a task is maintaining the top-k trending *hashtags* in Twitter, i.e., hashtags with the highest activity over a recent window in past. Another well-studied task is the computation of *global clustering coefficient* in presence of streaming updates to the network structure [69, 26]. A simpler aggregate query might be to continuously maintain, for all users, their friends that are (physically) closest to them (the aggregate function here is MIN). There are two key properties of aggregate functions that have significant impact on the computational complexity of the computation task: *duplicate sensitivity* and *decomposability*. A duplicate-insensitive aggregate function will return the same value even if some of its inputs are repeated. Examples include MAX, MIN, UNIQUE, etc. Duplicate-insensitive aggregates are amenable to additional optimizations during computation [80]. On the other hand, whether the aggregate function is *holistic* or decomposable has a significant impact on the optimizations that we can perform [80]. A holistic aggregate function (e.g., MEDIAN) requires all the input values to compute the final result, whereas decomposable aggregate functions are amenable to optimizations centered around partial aggregate computation and can be computed with much less memory. Clustering coefficient is an example of the latter type of aggregate function since the number of triangles can be counted (mostly) independently for each node.

5.2.5 Maintenance of Views or Other Derived Information

In this type of a task, the goal is to incrementally maintain the result of running an algorithm or performing a computation on the social network in presence of updates. Such tasks can be seen as a generalization of *materialized view maintenance* in traditional relational databases. In traditional view maintenance, the goal is to incrementally maintain the result of a declaratively-specified query; however, in social networks, the focus is often on more complex reasoning tasks. Examples of such tasks include incremental maintenance of *PageRank*, *dense subgraphs*, *spanning trees*, *shortest paths*, *communities*, and so on. In general, for any graph algorithm that is of interest in social network analysis, the question of incremental maintenance of the result in a dynamic setting may need to be addressed. For example, Bahmani et al. [23] address the problem of incrementally maintaining PageRank over a social network. Several works have considered considered the problem of incremental maintenance of dense subgraphs (e.g., [18]). The key challenge here is to avoid re-computation from scratch, and so far, most of the proposed techniques are heavily focused on a specific task.

5.3 Research Challenges and Future Directions

In this section, we look at the some of the key research challenges in supporting stream reasoning and querying tasks over social networks and briefly review the prior work on addressing those challenges. We stress that the area of stream reasoning over social network is still in its infancy, and the solutions discussed here should be considered as the starting point for future research on this topic.

5.3.1 Query Language

One of the major challenges in building general-purpose data management techniques or systems for stream reasoning over social networks is the lack of a high-level declarative query language for specifying the tasks. This issue arises in the context of graph data management in static settings as well. Well-established relational or XML query languages are not appropriate for graph-structured data because they lack support for specifying graph traversals. Although there have been proposals for graph query languages, none has gained wide acceptance; perhaps the only exception is the SPARQL query language, but the use of that query language has been largely limited to RDF datasets. This lack of a declarative language has led to a significant repetition of work by researchers that are developing tools for stream reasoning and querying over social networks. Clearly it is impossible to specify all of the wide range of tasks that we discussed in the previous section using a high-level, declarative language. However, we believe that it is possible to develop a declarative query language that will serve the needs of many stream reasoning and querying tasks; further, those tasks that cannot be fully expressed in the language can use the language to do part of the computation, with the remaining part done using a program written in a procedural language that ingests the result (analogous to how *user defined functions (UDFs)* are often used in conjunction with SQL in relational databases).

There are several starting points for designing such a query language. Several languages have been proposed in recent years that build upon SPARQL, e.g., Streaming SPARQL [33], Continuous SPARQL (C-SPARQL) [25], EP-SPARQL [19]. Although these languages focus on RDF data streams, they could be adapted to use in social networks by treating social network data as RDF data. Example 1 shows a C-SPARQL query that, given a stream of tweets along with the identified *hashtags* in it, returns all the hashtags with their cumulative frequencies within the last hour. Some of the key extensions to SPARQL include the use of "REGISTER QUERY" keyword to specify a continuous query that should be evaluated continuously, and a way to specify a window over the stream (using keyword "RANGE"),

Another option is to generalize XPath. For example, Mozafari et al. [90] propose XSeq, an extension to XPath to express both sequential and Kleene-closure expressions for XML streams. Example 2 shows an XSeq query that reports Twitter users who have been active for over a month. A key challenge here is that XPath

Example 1: *C-SPARQL Example [25]. Given the static user information and a stream of tweets, the query asks computes the total number of tweets per hashtag in last hour.*

```
1: REGISTER QUERY NumberOfTweetsPerHashTag COMPUTE
   EVERY 10m AS
2: PREFIX ex: <http://example/>
3: SELECT DISTINCT ?hashtag ?total
4: FROM STREAM <http://twitter.com/alltweets> [RANGE 1h STEP
   10m]
5: WHERE
6: ?user ex:from ?country .
7: ?user ex:tweets ?tweet .
8: ?tweet ex:has ?hashtag FILTER (?country="USA")
9: AGGREGATE { (?total. COUNT(?tweet). ?hashtag }
```

is designed to operate on tree-structured data, not graph-structured data. However, recent theoretical work suggests that it may be possible to use XPath for specifying graph queries [75].

Finally, the option that we have taken in our work [89, 86] is to extend Datalog [95] for this purpose. In recent years, Datalog has been shown to be an effective centerpiece in enabling declarative specification in a range of domains including networking [77], data cleaning [21], machine learning [36], and social network analysis [89, 98]. Compared to the above two languages, Datalog seems more amenable to be extended to support a large class of complex aggregate queries (e.g., global queries like *PageRank* computation, *shortest paths*, etc., can be specified using *recursion*). Datalog snippet in Example 3 specifies computation of *local clustering coefficient*, a measure of connectedness of a node's neighborhood. With some extensions, Datalog can also be used to specify social network transformation tasks as we showed in our prior work [89, 88]. Such flexibility may make a Datalog-based language a superior option in the end to specify a wide variety of stream reasoning tasks over social data.

Example 2: *XSeq Example: In a stream of tweets, report users who have been active over a month. A user is active if he posts at least a tweet every two days.*

```
1: return first(T)@userid
2: from /twitter/ Z* ($T)*
3: where tag(Z) = 'tweet' and tag(T) = 'tweet'
4: and T@date-prev(T)@date < 2
5: and last(T)@date-first(T)@date > 30
6: partition by /twitter/tweet@userid
```

Example 3: *Datalog Example [89]: Compute the clustering coefficient of each node.*

```
1: NeighborCluster(X, COUNT<Y, Z>) :=
2:                                  Edge(X,Y), Edge(X,Z), Edge(Y,Z)
3: Degree(X, COUNT<Y>) := Edge(X, Y)
4: ClusteringCoeff(X, C) :=
5:          NeighborCluster(X,N), Degree(X,D), C=2*N/D*(D-1)
```

5.3.2 *Efficient Execution Strategies*

Irrespective of how the stream reasoning tasks are specified, we must devise efficient execution strategies that can handle the very high update rates expected in online social networks. Below we briefly survey the key ideas that have been used successfully in past research on data streams for low-latency execution.

Incremental computation: The naive option of re-executing a query or a reasoning task when a new update arrives is likely to be infeasible except for very low rate data streams. Instead the goal of incremental computation is to maintain sufficient intermediate state in memory so that the new answer can be computed in an incremental fashion with minimal work. Such incremental techniques are unfortunately often specific to the task at hand. Eppstein et al. [47] is an early survey on the topic of dynamic graph algorithms. In recent work, Angel et al. [18] and Agrawal et al. [4] devise techniques for maintaining dense subgraphs; Bahmani et al. [23] present an approach to incremental computation of PageRank; Kutzkov et al. [73] present an incremental algorithm for computing clustering coefficient; Chandramouli et al. [38] explore real-time recommender system through incrementally maintaining their recommender model; and so on. A key research challenge here is to identify incremental techniques that are applicable to a wide variety of tasks (one way to do that is to focus on a high-level query language as we discussed in the previous section, e.g., C-SPARQL [24]). There is also often a natural trade-off between the amount of intermediate state that is maintained, and the amount of work that needs to be done when a new update arrives. Better understanding of this trade-off also presents a rich area for future work.

Sharing across multiple queries: Unlike traditional data management systems, in stream query processing systems, we may have thousands to millions of continuous queries running simultaneously. For instance, a personalized trend detection query where the goal is to monitor trends in every user's ego network can be seen as a collection of a large number of independent queries, one for each user. Sharing of computation across these queries is crucial in order to limit the computational cost. Such sharing has been shown to be an effective way to deal with high rate data streams in past work on data streaming systems [81, 45]. However, these types of techniques have not been well studied in social network setting. In a recent work, we designed novel techniques based on graph compression to exploit such sharing for continuous aggregate computation in social networks [86].

Approximate computation: One way to mitigate the execution complexity is to consider computing approximate answers instead of exact answers. This is especially attractive in scenarios where exact computation can be shown to be prohibitively expensive. For example, Becchetti et al. [27] show how one can incrementally compute local clustering coefficient with small error bounds where the exact algorithm [15] can require $O(n^{2.3727})$ time. Although there is much work on this topic in the data streams community, only recently have researchers started investigating similar problems for network algorithms. Zhao et al. [109] present a graph sketching technique, called gSketch, and show it can be used to answer several primitive frequency estimation techniques. Similarly, Ahn et al. [12] present graph sketching techniques for approximating *cut* values and for approximating the number of matches to a subgraph pattern query.

Sampling: Another general technique to deal with the high update rates is use random sampling to reduce the size of the data that needs to be processed. We may sample at two levels in a social network: first, we can try to sample from the network structure itself to reduce the size of the graph that needs to be processed; second, we can sample from the updates to the content. The latter is generally well-understood and the theory developed in the data streams literature could be extended for some types of queries. However, sampling the network structure is tricky since a naive random sample is likely to yield a network with very different properties than

the original network. We refer the reader to Ahmed et al. [9] for a detailed discussion of network sampling, both in static and streaming settings.

Parallel Computation: The increasing scale of most online social networks necessitates use of parallel and distributed solutions. Unfortunately computations on social networks are not easily distributable because of their highly interconnected nature. In fact, partitioning a social graph, which is key to distributed graph processing, is a hard problem to tackle because of overlapping community structure, and existence of highly connected dense components (cores) in most social networks. One of the simplest examples of a stream query on social data is a publish-subscribe query that asks to *fetch all updates from all friends* (this is also called *feed following*). Answering such queries with very low latencies is challenging if the data is distributed across a set of machines – for most users, their friends' data is likely to be located across multiple machines necessitating expensive distributed traversals. One extreme option is to replicate the data sufficiently so that, for each user, the required data (i.e., status updates of all their friends) is located on some machine [94]. However, both the memory overhead and the replica maintenance overhead can be very high for that solution [85]. More intelligent and sophisticated techniques for partitioning and replica maintenance must be developed to address these issues for more general stream reasoning and querying tasks. Another key challenge is designing appropriate distributed programming frameworks to support specifying general-purpose stream querying and reasoning tasks. Although there has been some progress on addressing this challenge in recent years (e.g., Kineograph [40], GraphInc [37]), much more needs to be done to scalably support a variety of complex stream querying and reasoning tasks.

6. CONCLUSIONS

Stream querying and reasoning over social data is an emerging research area that combines aspects from social network analysis, graph databases, and data streams, and is motivated by an increasing need for real-time processing of continuously generated social data. In this paper we presented a brief overview of this field, and discussed some of the key research challenges therein. There has been much work on specific problems in this field over the last few years. However, designing general-purpose data management systems that enable declarative specification of stream querying and reasoning tasks, and that can efficiently execute such tasks over high rate data streams, remains a fruitful direction for future research.

7. REFERENCES

[1] Neo4j open source NoSQL graph database. http://neo4j.org/.

[2] Trinity. http://research.microsoft.com/en-us/projects/trinity/.

[3] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, C. Erwin, Eduardo F. Galvez, M. Hatoun, Anurag Maskey, Alex Rasin, A. Singer, Michael Stonebraker, Nesime Tatbul, Ying Xing, R. Yan, and Stanley B. Zdonik. Aurora: A data stream management system. In *SIGMOD Conference*, page 666, 2003.

[4] Manoj K. Agarwal, Krithi Ramamritham, and Manish Bhide. Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments. *PVLDB*, 5(10):980–991, 2012.

[5] Charu Aggarwal, editor. *Data Streams: Models and Algorithms*. Springer, 2007.

[6] Charu Aggarwal and Haixun Wang. Graph data management and mining: A survey of algorithms and applications. In *Managing and Mining Graph Data*, pages 13–68. 2010.

[7] Charu Aggarwal, Yuchen Zhao, and Philip Yu. Outlier detection in graph streams. In *27th International Conference on Data Engineering (ICDE)*, pages 399–409, 2011.

[8] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *SIGMOD Conference*, pages 147–160, 2008.

[9] Nesreen K. Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network sampling: From static to streaming graphs. *CoRR*, abs/1211.3412, 2012.

[10] J. Ahn, C. Plaisant, and B. Shneiderman. A task taxonomy of network evolution analysis. *HCIL Technical Reports*, April 2011.

[11] Jae-wook Ahn, Catherine Plaisant, and Ben Shneiderman. A task taxonomy of network evolution analysis. *HCIL Tech Report, University of Maryland*, 2011.

[12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, 2012.

[13] Leman Akoglu and Christos Faloutsos. Anomaly, event, and fraud detection in large network datasets. In *WSDM*, 2013.

[14] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: spotting anomalies in weighted graphs. In *Proceedings of the 14th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 410–421, 2010.

[15] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 1997.

[16] Bernd Amann and Michel Scholl. Gram: a graph data model and query languages. In *Proceedings of the ACM conference on Hypertext*, pages 201–211. ACM, 1992.

[17] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 149–160, 2012.

[18] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *VLDB*, 2012.

[19] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, 2011.

[20] Apache. Giraph. http://incubator.apache.org/giraph.

[21] Arvind Arasu, Christopher Re, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, 2009.

[22] Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Trans. Knowl. Discov. Data*, 3(4):16:1–16:36, 2009.

[23] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proc. VLDB Endow.*, 4(3), December 2010.

[24] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri,

and Michael Grossniklaus. An execution environment for C-SPARQL queries. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 441–452, 2010.

[25] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *WWW*, 2009.

[26] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, 2008.

[27] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, 2008.

[28] Tanya Y. Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In *KDD*, 2006.

[29] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, 2002.

[30] Haixun Wang Bin Shao and Yatao Li. Trinity: A distributed graph engine on a memory cloud. In *SIGMOD*, 2013.

[31] Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4):175–308, 2006.

[32] Petko Bogdanov, Misael Mongiovì, and Ambuj K. Singh. Mining heavy subgraphs in time-evolving networks. In *ICDM*, pages 81–90, 2011.

[33] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming SPARQL: extending SPARQL to process data streams. *The Semantic Web: Research and Applications*, 2008.

[34] Lars Brenna, Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker M. White. Cayuga: a high-performance event processing engine. In *SIGMOD Conference*, pages 1100–1102, 2007.

[35] Matthias Brocheler, Andrea Pugliese, and V. S. Subrahmanian. COSI: cloud oriented subgraph identification in massive social networks. In *ASONAM*, 2010.

[36] Yingyi Bu, Vinayak R. Borkar, Michael J. Carey, Joshua Rosen, Neoklis Polyzotis, Tyson Condie, Markus Weimer, and Raghu Ramakrishnan. Scaling datalog for machine learning on big data. *CoRR*, abs/1203.0160, 2012.

[37] Zhuhua Cai, Dionysios Logothetis, and Georgos Siganos. Facilitating real-time graph mining. In *Proceedings of the fourth international workshop on Cloud data management*, CloudDB '12, pages 1–8, 2012.

[38] Badrish Chandramouli, Justin J Levandoski, Ahmed Eldawy, Mohamed F Mokbel, Big Data, and Justin Levandoski. Streamrec: a real-time recommender system. In *SIGMOD*, 2011.

[39] Badrish Chandramouli, Jun Yang, Pankaj K. Agarwal, Albert Yu, and Ying Zheng. Prosem: scalable wide-area publish/subscribe. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1315–1318, 2008.

[40] Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. Kineograph: taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 85–98, 2012.

[41] Sutanay Choudhury, Lawrence B. Holder, Abhik Ray, George Chin Jr., and John Feo. Continuous queries for multi-relational graphs. *CoRR*, abs/1209.2178, 2012.

[42] O. Cooper, A. Edakkunni, M. Franklin, W. Hong, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, and E. Wu. Hifi: A unified architecture for high fan-in systems. In *Proceedings of VLDB*, 2004. Demo.

[43] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, pages 412–422, 2007.

[44] Prasanna Kumar Desikan and Jaideep Srivastava. Mining temporally changing web usage graphs. In *WebKDD*, pages 1–17, 2004.

[45] Yanlei Diao, Peter Fischer, Michael J Franklin, and Raymond To. Yfilter: Efficient and scalable filtering of XML documents. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 341–342. IEEE, 2002.

[46] Yanlei Diao and Michael J. Franklin. Publish/subscribe over streams. In *Encyclopedia of Database Systems*, pages 2211–2216. 2009.

[47] David Eppstein, Zvi Galil, and Giuseppe F. Italiano. Dynamic graph algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999.

[48] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, June 2003.

[49] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 115–126, 2001.

[50] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, 2011.

[51] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: from intractable to polynomial time. *VLDB*, 2010.

[52] Huiji Gao, Jiliang Tang, and Huan Liu. Exploring social-historical ties on location-based social networks. In *ICWSM*, 2012.

[53] Jun Gao, Ruoming Jin, Jiashuai Zhou, Jeffrey Xu Yu, Xiao Jiang, and Tengjiao Wang. Relational approach for shortest path discovery over large graphs. *Proc. VLDB Endow.*, 5(4), 2011.

[54] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data-Stream Management – Processing High-Speed Data Streams*. Springer-Verlag, New York (Data-Centric Systems and Applications Series), 2011.

[55] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, 2008.

[56] GraphBase. Graphbase. http://www.graphbase.net/, 2011.

[57] D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In *ASONAM*, 2010.

[58] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and

Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 499–508, 2010.

[59] Ashish Gupta and Iderpal Singh Mumick. *Materialized views: techniques, implementations, and applications*. MIT press, 1999.

[60] R.H. Güting. GraphDB: Modeling and querying graphs in databases. In *Proc 20th Int. Conf. on Very Large Databases (VLDB)*, pages 297–308, 1994.

[61] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. SASE: complex event processing over streams (demo). In *CIDR*, pages 407–411, 2007.

[62] M. Gyssens, J. Paredaens, J. van den Bussche, and D. van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6:572–586, 1994.

[63] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Compact reachability labeling for graph-structured data. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 594–601, 2005.

[64] Huahai He and Ambuj K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *SIGMOD*, 2008.

[65] Jiewen Huang, Daniel J. Abadi, and Kun Ren. Scalable SPARQL Querying of Large RDF Graphs. *PVLDB*, 4(11):1123–1134, 2011.

[66] HyperGraphDB. A general purpose distributed data store, 2011. http://www.kobrix.com/hgdb.jsp.

[67] InfiniteGraph. Infinitegraph. http://www.infinitegraph.com/, 2011.

[68] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 595–608, 2008.

[69] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and Combinatorics*, volume 3595 of *Lecture Notes in Computer Science*, pages 710–716. Springer Berlin / Heidelberg, 2005.

[70] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[71] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. Pegasus: A peta-scale graph mining system. In *ICDM*, 2009.

[72] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *KDD*, pages 611–617, 2006.

[73] Konstantin Kutzkov and Rasmus Pagh. On the streaming complexity of computing local clustering coefficients. In *WSDM*, 2013.

[74] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, March 2007.

[75] Leonid Libkin, Wim Martens, and Domagoj Vrgoc. Querying Graph Databases with XPath. In *ICDT*, 2013.

[76] Zheng Liu, Jeffrey Xu Yu, Yiping Ke, Xuemin Lin, and Lei Chen. Spotting significant changing subgraphs in evolving graphs. In *ICDM*, pages 917–922, 2008.

[77] Boon Loo, Tyson Condie, Minos Garofalakis, David Gay, Joseph Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: language, execution and optimization. In *SIGMOD*, 2006.

[78] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. In *UAI*, pages 340–349, 2010.

[79] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning in the cloud. *PVLDB*, 5(8):716–727, 2012.

[80] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation service for Ad-Hoc sensor networks. In *OSDI*, 2002.

[81] Samuel Madden, Mehul A. Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *SIGMOD*, 2002.

[82] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *PODC*, 2009.

[83] Julian J. McAuley and Jure Leskovec. Discovering social circles in ego networks. *CoRR*, abs/1210.8182, 2012.

[84] I.A. McCulloh and K.M. Carley. Social network change detection. *Center for the Computational Analysis*, 2008.

[85] Jayanta Mondal and Amol Deshpande. Managing large dynamic graphs efficiently. In *SIGMOD*, 2012.

[86] Jayanta Mondal and Amol Deshpande. Supporting ego-centric aggregate queries over large dynamic graphs. Unpublished manuscript, 2013.

[87] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR '03: First Biennial Conference on Innovative Data Systems Research*. Asilomar, CA, 2003.

[88] Walaa Eldin Moustafa, Hui Miao, Amol Deshpande, and Lise Getoor. GrDB: a system for declarative and interactive analysis of noisy information networks: Demo. 2013.

[89] Walaa Eldin Moustafa, Galileo Namata, Amol Deshpande, and Lise Getoor. Declarative analysis of noisy information networks. In *ICDE GDM Workshop*, 2011.

[90] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. High-performance complex event processing over xml streams. In *SIGMOD*, 2012.

[91] S Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[92] Jeffrey Naughton, David DeWitt, David Maier, Ashraf Aboulnaga, Jianjun Chen, Leonidas Galanis, Jaewoo Kang, Rajasekar Krishnamurthy, Qiong Luo, Naveen Prakash, Ravishankar Ramamurthy, Jayvel Shanmugasundaram, Feng Tian, Kristin Tufte, Stratis Viglas, Yuan Wang, Chun Zhang, Bruce Jackson, Anurag Gupta, and Rushan Chen. The Niagara Internet query system. *IEEE Data Engineering Bulletin*, June 2001.

[93] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[94] Josep M Pujol, Vijay Erramilli, Georgos Siganos, Xiaoyuan Yang, Nikos Laoutaris, Parminder Chhabra, and Pablo Rodriguez. The little engine (s) that could: scaling online social networks. In *SIGCOMM*, 2010.

[95] Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of deductive database systems. *The Journal of Logic Programming*, 23(2):125–149, 1995.

[96] Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. On querying historial evolving graph sequences. In *VLDB*, 2011.

[97] John Scott. *Social network analysis*. SAGE Publications Limited, 2012.

[98] Jiwon Seo, Stephen Guo, and Monica Lam. Socialite: Datalog extensions for efficient social network analysis. In *ICDE*, 2013.

[99] Lei Sheng and Z.M. Ozsoyoglu. A graph query language and its query processing. In *Proceedings of the 15th International Conference on Data Engineering*, pages 572–581, 1999.

[100] Lei Tang, Huan Liu, Jianping Zhang, and Zohreh Nazeri. Community evolution in dynamic multi-mode networks. In *KDD*, 2008.

[101] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *KDD*, 2007.

[102] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, 2009.

[103] Silke Trissl and Ulf Leser. Fast and practical indexing and querying of very large graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 845–856, 2007.

[104] Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. A first step towards stream reasoning. In *FIS*, pages 72–81, 2008.

[105] Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[106] Fang Wei. Tedi: efficient shortest path query answering on graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 99–110, 2010.

[107] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD Conference*, pages 407–418, 2006.

[108] Hilmi Yildirim, Vineet Chaoji, and Mohammed Javeed Zaki. Grail: Scalable reachability index for large graphs. *VLDB*, 2010.

[109] Peixiang Zhao, Charu C Aggarwal, and Min Wang. gSketch: on query estimation in graph streams. *VLDB*, 2011.