Aggregate Estimation Over a Microblog Platform

Saravanan Thirumuruganathan[†], Nan Zhang[††], Vagelis Hristidis[‡], Gautam Das[†]
[†]University of Texas at Arlington [††]George Washington University [‡]University of California, Riverside
[†]{saravanan.thirumuruganathan@mavs, gdas@cse}.uta.edu [††]nzhang10@gwu.edu
[‡]vagelis@cs.ucr.edu

ABSTRACT

Microblogging platforms such as Twitter have experienced a phenomenal growth of popularity in recent years, making them attractive platforms for research in diverse fields from computer science to sociology. However, most microblogging platforms impose strict access restrictions (e.g., API rate limits) that prevent scientists with limited resources - e.g., who cannot afford microblog-data-access subscriptions offered by GNIP *et al.* - to leverage the wealth of microblogs for analytics. For example, Twitter allows only 180 queries per 15 minutes, and its search API only returns tweets posted within the last week. In this paper, we consider a novel problem of estimating aggregate queries over microblogs, e.g., "how many users mentioned the word 'privacy' in 2013?". We propose novel solutions exploiting the user-timeline information that is publicly available in most microblogging platforms. Theoretical analysis and extensive real-world experiments over Twitter, Google+ and Tumblr confirm the effectiveness of our proposed techniques.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.2.8 [**Database Management**]: Database Applications—*Data Mining*

Keywords

Aggregate Estimation; Microblogs; Twitter; Random Walk

1. INTRODUCTION

The Microblogs Query Aggregation Problem: Online microblogging platforms have experienced a phenomenal growth of popularity in recent years, because they offer easy and compelling ways for millions of users to post content and interact with each other. In addition to providing attractive mediums for person-person interactions, microblogging platforms also offer unprecedented opportunities for microblog data analytics, i.e., big-picture views of what people are saying, because they contain a deluge of opinions, viewpoints, and conversations by millions of users, at a scale that would be otherwise impossible to gather using more traditional methods such as controlled surveys. In fact, microblog service providers such as Twitter and their partners are already attempting to analyze their data, ranging from public opinion to spatiotemporal popularity of topics, and using the results to build advertising campaigns or monitor the reputation of companies.

Although these are important applications for companies, microblogging platforms also provide free (but limited) public access to their data in the form of restricted APIs, which offer great opportunities for other, often non-commercial applications, such as the type of studies that would be most useful to a social scientist. For example, a social researcher may wish to analyze publicly available microblog conversations and postings to determine the change in general public's attitudes on individual privacy before and after the news of Edward Snowden's leakage of NSA surveillance became public. Other examples can include studies of the spread of obesity-promoting attitudes, the mechanisms of bullying in colleges or schools, and the early detection of suicidal discourse.

A core functionality to facilitate such analytics is to answer *aggregate queries* over publicly available microblog data, which is the focus of this paper. An example of aggregate query is "How many Twitter users used the keyword `privacy` in 2013?". We shall consider SUM, COUNT, AVG queries on various attributes of microblog users or posts (e.g., users' age or posts' length), with selection conditions on keywords and other attributes like time.

We emphasize that our techniques will necessarily generate approximate answers; exact answers are infeasible since they require access to the complete data (and are also often unnecessary in many applications, since approximate aggregates are usually sufficient for obtaining "big-picture" views of the data). Our methods should be efficient in the following sense - the number of API calls made to the microblogging service provider should be as few as possible in generating the approximate aggregate.

Limitations of Existing Microblog APIs: Many of the popular microblog sites like Twitter, Tumblr, Instagram, Yammer, Weibo, Identi.ca (and some other social networking sites like Google+ and Facebook that also offer microblogging features) offer search API calls, which allow retrieving posts containing query keywords, but the results are limited, e.g., past week in Twitter Search API [6]. Other microblogs limit the maximum number of search results one could retrieve to at most a few thousands.

A notable exception to such search APIs is Twitter's Streaming API, which allows retrieving large numbers of posts given keyword and other conditions[1]. Unfortunately, the streaming interface only allows retrieving Twitter postings *in the future*, and there is no way to obtain historical tweets. Thus, a a sociologist will never be able to study the origin of a trending keyword unless he/she is somehow

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2610517 .

[1]If no condition is specified, the streaming API returns a $\sim$1% sample of all tweets - a ratio too small to reliably compute many aggregates (e.g., those that are conditioned on a keyword).

(magically) able to predict such a keyword ahead of time. Note that there are companies like GNIP [2] and Datasift [1] that sell historic microblog data; however the subscription fee is often rather high (e.g., \$3,000 per month for Twitter alone at Datasift.com [1]) for a non-commercial setting such as social science research.

Limitations of Previous Research on Estimation of Aggregates on Social Networks: There has been work on estimating aggregate functions on social networks [13, 15, 17, 24]. These works generally use random walk-based sampling on the social graph, or adaptations of it like Metropolis-Hastings [12]. However, they are inefficient for the type of aggregate queries that we study for the following reasons: They only consider *broad aggregates*, that is, aggregates on the whole social network, and not constrained by keywords. Most of these techniques enable aggregate estimation by drawing a random sample of *all* microblog users, and extrapolating from the sample. For our purpose, however, aggregate queries have keyword selection conditions that match only an extremely small fraction of these users - e.g., the number of Twitter users who have used the keyword `privacy` in their postings is only 0.4% of all active users. A straighforward solution would be to only consider users who satisfy the selection condition during the sampling random walk. However, we found that this leads to a social subgraph with tightly connected communities that that significantly increase its convergence time (its *burn-in* period).

Outline of Our Results: We develop MICROBLOG-ANALYZER, an efficient platform to enable the accurate estimation of aggregate queries over an online microblogging service. Its design is based on a central and novel idea: to leverage the *user-timeline interface* (offered by most microblogging platforms) to bypass the above-described limitations on the search API. The user-timeline API inputs a user-id and returns all (for all practical purposes as discussed in §2) public posts generated by the user.

MICROBLOG-ANALYZER operates as follows: we start from a user who recently generated a post satisfying the aggregate query keyword condition (e.g., who is returned by the search API), and then traverse a carefully constructed *subgraph of the social graph*, where users are nodes and user connections are edges, according to the aggregate query, in order to sample (and retrieve through the user-timeline interface) a small number of user timelines based on which we generate our aggregate estimation. There are two main technical issues facing this design: (1) how to design the aggregate-dependent subgraph, and (2) how to traverse such a subgraph, in order to enable efficient and accurate aggregate estimations.

First, we propose a *level-by-level subgraph* to address the issue of subgraph design. Specifically, we introduce a novel taxonomy of user connections (i.e., edges) based on the aggregate being estimated and user timelines. A critical feature of this taxonomy is our finding that, while certain types of edges are beneficial to efficient sampling, others are detrimental to it and should be *removed* from the graph. We adjust the original social graph according to this taxonomy to produce the level-by-level subgraph and, by performing simple random walks [20] over it, develop MICROBLOG-ANALYZER-Simple Random Walk (MA-SRW), our first algorithm for aggregate estimations over a microblog platform. We present theoretical analysis and real-world experiments to demonstrate the superiority of MA-SRW over several baseline graph designs.

Then, to address the graph traversal issue, we develop a *topology aware random walk* over the level-by-level subgraph. Previous random walk techniques (e.g., as used in MA-SRW) are oblivious and therefore generic to the topology of the underlying graph. This often requires a large query cost for the so-called burn-in period [12] in order for the sampling probability of each node to converge to a stationary distribution, so that the sampled nodes can be used for aggregate estimations. We show that, by leveraging knowledge of the underlying graph topology - specifically, the level-by-level structure - our traversal algorithm removes the need of this burn-in period (and the associated query cost) - enabling a significantly more efficient and accurate aggregate estimation process. The execution of topology-aware random walk over the level-by-level graph forms our final algorithm, MICROBLOG-ANALYZER-Topology-Aware Random Walk (MA-TARW).

Summary of Contributions:

- We define the novel problem of aggregate estimation over historic microblog data (§2). We develop a novel idea leveraging the user-timeline access provided by online microblogs to bypass the limitations they place on the search API, and present a platform to tackle the aggregate estimation problem (§3).
- To effectively sample the social graph according to an aggregate query, we develop a level-by-level subgraph topology and demonstrate through theoretical analysis and experimental results its superiority over a number of baseline graph designs (§4).
- To efficiently sample a level-by-level graph, we develop a topology aware random walk which leverages the special properties of a level-by-level graph topology to significantly outperform baseline solutions such as traditional random walks (§5).
- We present comprehensive experiments on Twitter, Google+ and Tumblr that show the significant improvement our methods offer compared with the state-of-the-art (§6).

2. PROBLEM DEFINITION

In this section, we start with describing a data-access model that abstracts the API interfaces provided by most popular microblogs, and then define the problem of aggregate estimation.

Model of Microblog Data Access: In general, a microblogging platform offers three functionalities: (1) share concise updates in text (e.g., Twitter, Google+, Tumblr), image (e.g., Instagram), or video (e.g., Vine); (2) form social connections with each other (e.g., follower/followee in Twitter, Circles in Google+, Likes in Tumblr); and (3) search, subscribe to, and consume the updates posted by users. Correspondingly to these three functionalities, most microblogging platforms - e.g., Twitter, Tumblr, Instagram, Google+, Weibo, Yammer etc. allow the following three types of *queries*:

1. `SEARCH`: Given a keyword (or keywords) w, return *recent* microposts that contain w. Most microblog sites only return posts in recent weeks – e.g., the last weeks posts in Twitter API [6]. Other microblogs restrict search to top-k results where k could be in the low thousands. They do so for two main reasons: recent data are generally more interesting to users, and many microblog service providers consider selling access to historic data an important monetization channel [1, 2].
2. `USER CONNECTIONS`: Given a user u, return all other users "connected" with u. Note that "connections" here are loosely defined - they can be follower/followee relationships (as in Twitter), friendships (as in Friendfeed), etc. Almost all real-world microblogs, e.g., Twitter, Instagram, Tumblr, allow complete access to all user connections (unless a user sets it to private).
3. `USER TIMELINE`: Given a user u, return all posts published by u. To simplify the taxonomy, we assume that a user timeline query also returns the user's profile information (e.g., name, demographics). Like user connection queries, real-world microblogs seldomly limit the returned user timelines, with one

notable exception of Twitter which only publishes the most recent 3200 tweets published by a user. Nonetheless, according to recent studies, only a very small fraction of extremely prolific users - 5% [4] - have posted more that 3,200 tweets and even for these users only very old tweets are missing, in contrast to the search API that only goes back one week [6] (e.g., even Justin Bieber only posted 2,500 tweets between Apr and Dec 2013). Given that in this paper we focus on aggregate estimations, it is safe to assume that this small number of incomplete user timelines has little effect on the estimated aggregates.

Note that the above interfaces could alternatively be implemented through *web crawling* of the microblog site if an API is not available. However, web search interfaces often have unknown selection and ranking criteria that make them less desirable for aggregate estimations - e.g., in Twitter, posts may be missing from the web search but not from the search API results [6]; similarly Tumblr and other sites often perform unpredictable query expansion at their Web search interface. Further, many sites do not allow web-page scraping, e.g., as specified in Twitter (https://twitter.com/tos).

Another important interface limitation imposed by microblogging APIs is an upper bound on the number of queries a user can issue in a time period. For example, Twitter's search API [6] allows only 180 queries over a 15 minute window, and Reddit API allows no more than one request every two seconds.

Problem Definition: In this paper, we address the problem of aggregate estimations over microblogs by issuing queries through the above-described limited microblog interface. Specifically, we consider aggregate queries of the form `SELECT AGGR(`$f(u)$`) FROM` U `WHERE CONDITION` where U is the set of all users, $f(u)$ is any function that returns a numeric measure for each user u (e.g., age or #connections), `AGGR` is an aggregate function such as COUNT, SUM or AVG, and `CONDITION` determines whether a user$u$ should be considered for (i.e., included in) the aggregate.

It is important to note that the above-described form covers not only aggregates over users, but also *aggregates over posts* as well. For example, the COUNT of posts containing keyword `privacy` can be specified as follows: `CONDITION` returns TRUE if a user has `privacy` appearing in its timeline, and FALSE otherwise; $f(u)$ returns the number of posts containing `privacy` in the user's timeline; and `AGGR` is SUM.

While many different predicates can be specified in `CONDITION`, we highlight two specific types: (a) *keyword predicates* - i.e., a user is included iff its timeline contains a pre-determined keyword (e.g., `privacy` in the above example); (b) *time window* - e.g., users who mentioned `privacy` from Jul to Dec 2013. Keyword predicates are prevalent in aggregates required by social science studies because most of these studies focus on one or a few topics specifiable as keywords. For this reason, in this paper we focus on aggregate queries with at least one keyword predicate, optionally a time window, as well as other other predicates on a user's profile attributes (e.g., gender, age, number of connections).

Performance Measures: The performance of an aggregate estimation algorithm is measured in terms of efficiency and accuracy. Given the query-rate limit enforced by most microblogging platforms, the efficiency is the *query cost* - i.e., the number of queries and/or API calls (on `SEARCH`, `USER CONNECTIONS`, and `USER TIMELINE`) the algorithm issues to the microblog.

For accuracy, given estimation $\tilde{\theta}$ of an aggregate θ, we apply the standard measure of *relative error* $|\tilde{\theta} - \theta|/\theta$. Note that the error is determined by two factors[2]: *bias*, i.e. $E(\tilde{\theta} - \theta)$, and *variance* of $\tilde{\theta}$.

[2]Specifically, the mean squared error MSE = bias^2 + variance.

Hence, given an aggregate query with keyword and other predicates, the objective of the *microblog aggregate estimation problem* studied in this paper is to produce an estimation while minimizing both query cost and relative error.

3. OVERVIEW OF MICROBLOG-ANALYZER

This section overviews MICROBLOG-ANALYZER, our system for enabling analytics over a microblog by issuing queries through its limited access interface. We start by presenting a key idea of MICROBLOG-ANALYZER: estimating aggregates by sampling user timelines. Then, we outline the design issues associated with two main components of MICROBLOG-ANALYZER: (1) GRAPH-BUILDER, i.e., the generation of a conceptual graph that connects user timelines together, and (2) GRAPH-WALKER, i.e., the design of an efficient sampling algorithm over such a graph. While §4 and §5 describe these two components in detail, we discuss at the end of this section how we prototyped MICROBLOG-ANALYZER over Twitter and collected ground-truth for its evaluation.

3.1 System Architecture

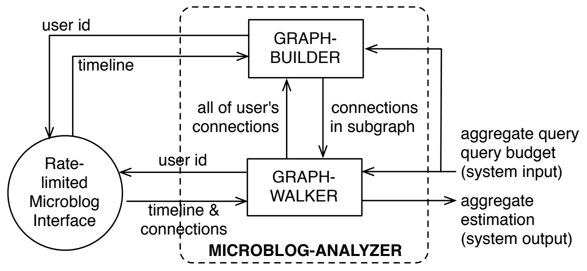

Figure 1: System Architecture

Figure 1 depicts the architecture for MICROBLOG-ANALYZER which has two main components: (1) GRAPH-BUILDER that builds a graph connecting users and (2) GRAPH-WALKER that performs a random walk over such a graph. The system works as follows:

- It receives as input an aggregate query to be estimated (as defined in §2), a query budget (i.e., the maximum number of queries MICROBLOG-ANALYZER can issue to the microblog), as well as one or a few "seed users" which have posted microblogs satisfying the selection condition of the aggregate. Note that such seed users can be easily identified through the limited search API (e.g., for Twitter, users who posted a keyword in the past week).
- Given a seed user, MICROBLOG-ANALYZER uses the GRAPH-BUILDER to determine which other users are its *neighbors*. As we shall show later, the design of GRAPH-BUILDER can range from simply using all social connections of the user to a carefully designed algorithm that takes into account the aggregate being estimated and certain user timeline information to select a subset of such social connections. We shall discuss the design of this component in the next subsection and then in detail in §4.
- Given the neighbors, GRAPH-WALKER determines the probability for MICROBLOG-ANALYZER to "transit to" and sample each neighbor for aggregate estimation. Once again, the design ranges from simply choosing each neighbor uniformly at random (i.e., simple random walk) to a carefully designed algorithm that takes into account certain topological properties of the graph produced by GRAPH-BUILDER. We shall discuss the design of this component in the next subsection and then in detail in §5.

Table 1: Components employed by proposed algorithms

	GRAPH-BUILDER	GRAPH-WALKER
MA-SRW	Level-by-Level (§4)	Level-by-Level (§4)
MA-TARW	Simple RW [20]	Topology-Aware RW (§5)

- The above process can be repeated multiple times until exhausting the query budget, so as to produce a more accurate aggregate estimation as the final output of MICROBLOG-ANALYZER.

3.2 Key Idea: User-Timeline Based Analytics

Feasibility of User-Timeline Based Analytics: To address the often stringent limit on search query interfaces, a key data source MICROBLOG-ANALYZER leverages is the user timeline - i.e., all historic posts published by a user - which, as discussed in §2, is readily accessible through the access interface of many microblogs.

To understand why user-timeline information can be used to answer aggregate queries (especially those with keyword predicates) defined in §2, we start by considering an extremely inefficient technique which nevertheless demonstrates the feasibility of this idea. Note that, as shown by many previous studies [13, 18, 24], the vast majority of users in a microblogging service are linked in a connected graph through social relationships revealed by the service - e.g., follower/followee in Twitter, Circles in Google+, blog followers in Tumblr, comments on same post in Reddit, etc. For the purpose of this paper, we consider such a *social graph* to be undirected. For directed relationships such as follower/followee on Twitter, one can easily convert them to undirected edges by considering two users to be connected if either follows the other.

Given the social graph, one can simply start with one user and recursively follow edges (using user connections API) to reach and crawl the timeline of most users - making it possible to answer aggregates based on the locally crawled data. While this brute-force method demonstrates the feasibility of acquiring sufficient information (for aggregate estimation) through user-timeline queries, it unfortunately requires a prohibitively high query cost given the access-rate limit discussed in §2. In addition, most crawled data would be completely useless for aggregate estimation - e.g., even for a broad query like the count of users who have tweeted `privacy` in 2013, the vast majority of user timelines would be irrelevant because only a very small percentage ($\approx$ 0.4% of its active users) of all Twitter users satisfies the selection condition - leading to a significant waste of resources.

To address this problem, MICROBLOG-ANALYZER only *samples* users who satisfy the keyword predicate specified in the aggregate query, and then produce aggregate estimations according to the collected sample. There are two design issues that are critical for enabling the sampling-based method:

Design Issue 1 (Subgraph Generation): A straightforward method to sample user timelines is to perform a *random walk* over the social graph - e.g., a simple random walk [20] recursively jumps from one user to one of its neighbors chosen uniformly at random - so timelines of sample users (taken after a sufficient number of "burn-in" transitions [12]) can be used for aggregate estimations. A problem with this method, however, is that topology of the social graph is very "*unfriendly*" for sampling and requires a high query cost for random walks to "burn-in". While we shall discuss this finding in detail in §4, an intuitive explanation here is that the social graph contains many "redundant" edges which may "trap" a random walk inside a tightly connected component - i.e., preventing the walk from efficiently sampling all nodes in the graph.

As such, to enable efficient sampling, the first design issue we must address is how to "on-the-fly" remove the redundant edges and find a *subgraph* that satisfies two conditions: (1) *high recall:* it still includes most if not all users who satisfy the selection condition of the aggregate to be estimated, and (2) *sampling-friendly:* the subgraph should have a "well-knit" [20, 26] topology and therefore facilitate an efficient random walk process. One can see that the high-recall requirement ensures the closeness of estimations generated from the subgraph to the ground truth, while the friendliness requirement ensures an efficient random walk process. We shall develop a novel technique for subgraph construction in §4.

Design Issue 2: Sampling Design: In the above discussions, we considered a direct application of traditional random walk techniques (e.g., simple random walk [20] or Metropolis-Hastings random walk [12]) over the user-timeline graph (or subgraph, once the above design issue is addressed). While there has been a large body of work on using these random walks for aggregate estimation over large graphs [13, 15, 17, 19, 20] a key deficiency of it is the significant query cost required by answering COUNT and SUM queries.

While samples collected by random walks can be directly used to estimate AVG queries (as a weighted average of all sample tuples), if one does not know the total number of nodes in the graph (which is often the case in practice), generating estimations for COUNT and SUM often needs to use a significantly more expensive mark-and-recapture [9] based technique (e.g., [15]). However, in this method $\Omega(\sqrt{n})$ samples are needed to produce just one collision over an n-node graph - an extremely high query cost even for a perfectly built subgraph containing only users satisfying the selection condition. For example, to estimate the COUNT of all users who tweeted `privacy` in 2013 (about 894,000), this means at least thousands of samples must be collected, incurring a very high query cost. To address this deficiency, the second design issue is how to efficiently traverse the graph to estimate AVG, COUNT and SUM aggregates. We shall develop a novel sampling algorithm to achieve these objectives in §5. Table 1 shows which subgraph generation (GRAPH-BUILDER) and graph sampling (GRAPH-WALKER) components are employed by the two key proposed algorithms of this paper.

Prototype Design for Twitter Experiments: Before presenting out detailed design of MICROBLOG-ANALYZER in §4 and §5, we would like to briefly discuss how we prototyped over Twitter, the preeminent micro-blogging platform. Note that while we focus the rest of the paper on this Twitter prototype, the adaption to other micro-blogging platforms is straightforward - e.g., we present experiments in §6 over Google+ and Tumblr.

Twitter's REST API [5] naturally fits into the data access model detailed in §2. The search API retrieves tweets matching the given keywords which were posted during the past week [6]. The user timeline API provides access to a user's historic tweets (up to the last 3200). Since Twitter allows asymmetric relationship between users, we have to use two APIs to retrieve all the users who follow user u and all users who are followed by u, in order to collect all user connections as defined in the undirected social graph. Each API call returns up to 5000 connections while the vast majority (upwards of 95% [4]) of users have fewer than 100.

We now briefly describe how we collected the ground truth for evaluating our prototype's effectiveness on estimating aggregates such as "COUNT of all users who tweeted about `privacy` in 2013". We used the streaming API to collect all public tweets mentioning a diverse set of keywords (such as cities, celebrities, organizations, etc.) between Jan 1, 2013 to Oct 31, 2013. Twitter ensures that the stream returns all relevant tweets as long as their frequency is less than about 1% of the entire Twitter Firehose (total volume) [7]. Our specified keywords were selective and did not receive any rate limit exception, which means that this is an accurate ground truth to evaluate aggregate estimation algorithms.

4. LEVEL-BY-LEVEL SUBGRAPH

Recall that GRAPH-BUILDER aims to construct a subgraph (of the social graph defined in §3) with two properties: (1) a high recall of (timelines of) users who satisfy the selection condition of the aggregate query to be estimated, and (2) a topology that enables efficient sampling of such users. In this section, we start by describing a baseline method that achieves (1) but fails (2). The deficiencies of this baseline motivate us to propose a novel *level-by-level* subgraph to satisfy both. At the end of this section, we present Algorithm MA-SRW which enables aggregate estimation by performing simple random walks over the level-by-level graph.

Running example: Throughout this section and the next, we consider as running example the estimation of the following aggregate query over our Twitter prototype: AVG(number of followers) of users who tweeted the keyword `privacy` in 2013.

4.1 Baseline Subgraphs and Their Deficiencies

We start with discussing *term-induced subgraph*, a straightforward subgraph construction which serves as a baseline for our study. Simply put, unlike the original graph which includes all user timelines, the term-induced graph consists of only users who satisfy the *keyword selection condition* of the aggregate query. In the running example, this leads to a subgraph consisting of all users who have tweeted `privacy` before. From a practical standpoint, this means that, during the random walk process, we always start with a user who has `privacy` in his/her timeline and only transit to users who satisfy the same criteria.

The rationale for this baseline approach is simple: Since nodes in the term-induced subgraph form a superset of those covered by the aggregate, the subgraph has a high recall as long as it remains connected or has a large connected component. On the other hand, the sampling efficiency is likely to be improved because of the reduced graph size. The design of the subgraph balances between the two objectives by filtering nodes only with keyword predicates (defined in §2) - which, as shown below, vastly reduces the subgraph size while keeping it connected - but not other conditions in the aggregate query - e.g., a time-interval condition which, when overly short, can result in a low recall.

Our experiments on Twitter confirmed the validity for the high-recall assumption - for all keywords and hashtags we tested (from popular ones such as `Fiscalcliff, New York, Superbowl` to more obscure ones such as `Tunisia`, `Simvastatin`), the largest connected component of the subgraph contains almost all (on average 94% - see Table 2 for details) nodes in the subgraph - demonstrating the high-recall of a term-induced subgraph. Intuitively, this is because of the strong correlation between social relationships and co-mentioning of keywords - i.e., not only are terms/hashtags likely propagated between followers and followees, but users who have similar interests tend to be connected *and* use the same keywords - leading to the high recall.

For sampling efficiency, our findings were mixed. While the query cost is indeed much lower than the original social graph, it is still very expensive. For the running example (average number of followers for users who tweeted `privacy`), this subgraph required close to 49,000 queries to obtain an estimate with less than 5% relative error. While this value is significantly less than than the 144,000 queries required for the original graph, it is still high con-

Table 2: Statistics: Term Induced & Level-by-Level Subgraphs

Keyword	Recall	Avg#common neighbors	% of intra & cross-level
`FiscalCliff`	97%	16, 2	27%, 1%
`New York`	91%	49, 3	32%, 2%
`Super Bowl`	93%	34, 1	29%, 2%
`Obamacare`	96%	21, 5	22%, 1%
`Tunisia`	86%	11, 4	28%, 1%
`Simvastatin`	81%	19, 2	24%, 2%
`Oprah Winfrey`	91%	22, 4	29%, 3%

sidering Twitter's rate limit. Figures 2 and 3 how the term-induced subgraph performs on estimating AVG(number of followers) and COUNT for users who tweeted `privacy`, respectively.

To understand why the efficiency problem remains with the term-induced subgraph, we note that even though users who tweeted `privacy` only represent a small percentage of all Twitter users, the number of *edges* connecting them in the term induced graph is still very large (e.g., close to 1 million edges connecting approximately 142 thousand nodes for the running example). With such a large and dense graph, the efficiency of sampling critically depends on whether the graph topology is carefully designed to enable efficient random walks.

Unfortunately, we found a special topological property of the term induced subgraph that is indeed very "unfriendly" for efficient sampling: Note that, exactly because of the same reason why the term-induced graph likely has a high recall, keywords are often propagated among users that form tightly connected communities (e.g., measured according to graph modularity [26]). This actually requires a random walk to have a long burn-in period because it is likely "trapped" inside a tightly connected community before having a sufficient probability to propagate to other parts of the graph. Our experiments on Twitter confirmed this finding. The burn-in period (with Geweke threshold [11] $Z <= 0.1$) for the entire Twitter graph and the term induced subgraph (for `privacy`) were approximately 700 and 610 respectively. Similar behavior was observed for other keywords also (see Figure 4 for details).

One can see from the above discussion that the straightforward design of a term-induced subgraph cannot adequately address the sampling-efficiency problem of the original social graph, mainly because of the long burn-in dictated by traversing between tightly connected communities. In the next subsection, we describe our proposed methods for constructing a "sampling-friendlier" subgraph topology - specifically, by exploiting *time dimension* of the term-induced subgraph - i.e., the time order with which users posted a specified term like `privacy`.

4.2 Level-by-Level Subgraph

4.2.1 Key Idea and Rationale

To develop our idea of a level-by-level subgraph, we start with introducing a taxonomy of edges in the term-induced subgraph and discuss how each type of edges affect the efficiency of random walks. Consider a simple organization of all nodes (users) into multiple *levels* according to the time when a user first qualified for the keyword predicate (i.e., tweeted `privacy` in the running example). Consider an arbitrary time interval, say 1 day. We partition all users in the term-induced subgraph into multiple segments according to the interval (e.g., users published `privacy` between Jan 1, 13 and Oct 31, 13 will be partitioned into 303 segments).

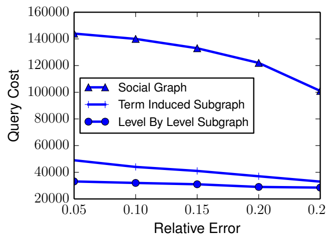

Figure 2: AVG(followers): Users who tweeted privacy

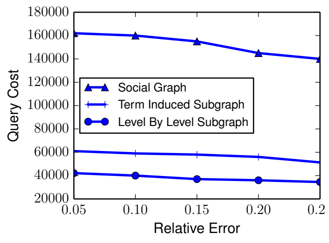

Figure 3: COUNT: Users who tweeted privacy

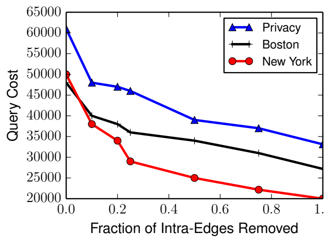

Figure 4: Impact of removing intra-edges on Query Cost

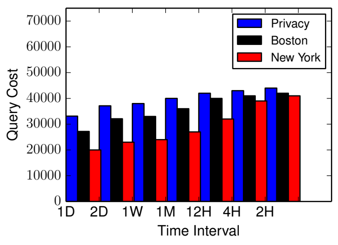

Figure 5: Impact of T on query cost (H=hours, D=days, W=weeks, M=months)

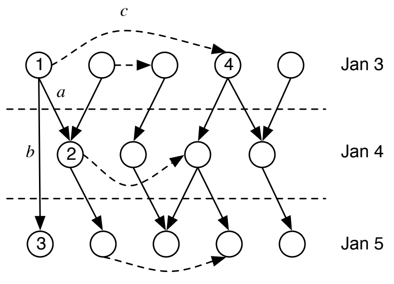

Figure 6: Level-by-Level View of term-induced Subgraph H.

If we draw each segment as a "virtual level" as in Figure 6, and place these levels from top to bottom in chronological order, then we can classify all edges in the subgraph into three categories: *(a) Adjacent-level edges* connect two users in adjacent levels - e.g., Edge a in Figure 6 connects User 1 who first tweeted `privacy` on Jan 3 and User 2 who did so on Jan 4. *(b) Cross-level Edges* connect two users in unequal and non-adjacent levels - e.g., Edge b in Figure 6. *(c) Intra-level Edges* connect two users in the same level - e.g., Edge c in Figure 6.

The reason why we introduce such a ternary classification is because, interestingly, these three types of edges serve different roles in facilitating or deterring the random walk process. Specifically, we found that, for a "reasonable" time interval (>1 hour), (more) intra-level edges are *detrimental* to the efficiency of random walks, while (more) adjacent-level edges are *beneficial* to it. Cross-level edges, on the other hand, contribute to more efficient random walks but are relatively rare in practice (e.g., less than 1% for `privacy`. See Table 2 for other keywords).

While we shall verify this finding both theoretically and experimentally in §4.2.2, we would like to start here with an intuitive explanation for the varying effects different types of edges have on sampling efficiency. Intuitively, intra-level edges usually exist between users in a tightly connected component (as described in §4.1), while adjacent- and cross-level edges are most often not. This has been observed before - e.g., it was found in [3] that 92% retweets produced by followers of a user occur within 1-hour of the original tweet, demonstrating that most followers "respond" within very short time, leading to intra-level edges between users in a tightly connected component. Our experiments confirmed this observation - e.g., Table 2 (column 2) contrasts, for various keywords, the average number of common neighbors shared by two users connected by an intra-level edge and those who are not. We can observe that, on average, one in four edges in the term induced subgraph is an intra-level edge. Further, the users connected by an intra level edges have significantly more common neighbors.

One can see from this explanation that, to "burn-in" to a stationary distribution, a random walk needs to cross adjacent- and/or cross-level edges and cannot get "stuck" inside a small group of users tightly connected by intra-level edges. Combine this with the fact that a substantial percentage of edges in a real-world term-induced graph are intra-level ones (e.g., even for a short interval of 1 hour, more than 28% of edges for keyword `privacy` are intra-level ones), a key idea for our subgraph design subgraph is to *remove all intra-level edges* from the term-induced graph. We refer to this subgraph as the *level-by-level subgraph* because it only contains edges between different levels. From a practical standpoint, this means that the random walk needs to follow a simple rule: transit from a user to its neighbor if and only if they did not first tweet `privacy` in the same day.

One can see that, to properly design a level-by-level subgraph, one needs to address two key issues. One is, of course, to verify that removing intra-level edges indeed improves the efficiency of random walks. We shall discuss this verification in §4.2.2. The other has to deal with the *time interval* used in defining intra-level edges. Note that an intra-level edge could be classified as adjacent- or even cross-level edge with a different time interval. We shall develop the proper setup of time interval for an aggregate in §4.2.3.

4.2.2 Effect of Intra-Level Edges

We now analyze the effect of intra-level edges on the efficiency of random walks in two steps. First, we present theoretical analysis on a simple example of level-by-level graph to illustrate how the removal of intra-level edges makes the graph more "well-knit" and more efficient for random walks. Then, we present experimental findings from our Twitter prototype to demonstrate the efficiency improvements achieved by removing intra-level edges.

For theoretical analysis, we consider the change of *graph conductance* [20] after the removal of intra-level edges. The *conductance* $\varphi(G)$ of a graph G measures how "well-knit" G is - i.e., how fast a random walk can converge to its stationary distribution. Specifically, we have

$$\varphi(G) = \min_{S \subseteq V} \frac{\sum_{v_i \in S, v_j \in \overline{S}} a_{ij}}{\min\left(a(S), a(\overline{S})\right)} \tag{1}$$

where V is the set of vertices in G, S and $\overline{S} = V \backslash S$ form a partition of V into two disjoint subsets, $a_{ij} = 1$ if there is an edge connecting v_i and v_j in G and 0 otherwise, and $a(S) = \sum_{v_i \in S} \sum_{v_j \in V} a_{ij}$. In general, a simple random walk burns-in faster on graphs with higher conductance [21].

Given the complexity of analyzing the conductance of an arbitrary graph, for the purpose of this paper, we consider a simple example of a level-by-level subgraph G as follows. Let there be n nodes in the graph which are distributed evenly across h levels (so each level contains n/h nodes). The adjacent-level edges in the graph are constructed such that each node at Level i ($i \in [1, h-1]$) is connected with d nodes chosen uniformly at random from those at Level $i+1$. The intra-level edges, on the other hand, connect each node at Level i with d' other nodes chosen uniformly at random from Level i. While this simple model does not match real-world graph topologies, it nevertheless gives us an indication of how intra-level edges affect conductance, as demonstrated in the following theorem.

THEOREM 4.1. *The conductance for G is*

$$\varphi(G) = \begin{cases} \frac{h}{(k+d)(h-1)n} & \text{if } d \le \frac{n}{2h}, k \le \frac{n}{2h} \\ \min\left(\frac{2kh-n}{kh+dn}, \frac{2d}{2d(h-1)+hk}\right) & \\ & \text{if } d \le \frac{n}{2h}, \frac{n}{2h} < k < \frac{n}{h} \\ \min\left(\frac{2dh-n}{kh+dn}, \frac{2d}{2d(h-1)+hk}\right) & \\ & \text{if } \frac{n}{2h} < d < \frac{n}{h}, k \le \frac{n}{2h} \\ \min\left((k-\frac{n}{2h})\frac{2dh-n}{kh+dn}, \frac{2d}{2d(h-1)+hk}\right) & \\ & \text{if } \frac{n}{2h} < d < \frac{n}{h}, \frac{n}{2h} < k < \frac{n}{h} \end{cases} \tag{2}$$

After removing all intra-level edges, the conductance of G' is

$$\varphi(G') = \begin{cases} \frac{h}{nd(h-1)} & \text{if } d \le \frac{n}{2h} \\ \min\left(\frac{2hd-n}{nd}, \frac{1}{h-1}\right) & \text{if } \frac{n}{2h} < d < \frac{n}{h} \end{cases} \tag{3}$$

PROOF. We give a proof sketch due to space limitations by showing *adding* intra level edges to a level by level graph actually decreases conductance. For simplicity, we consider a level-by-level graph (G') with h levels where each level has exactly n/h nodes. Each node is connected with $d(d \ll n/h)$ randomly chosen nodes in adjacent levels. In order to compute the conductance of this graph, we have to identify the cut that has the lowest conductance. There are two possible cuts - horizontal (where the cut disconnected two adjacent levels) or vertical (where the cut disconnects the graph into subgraphs, each with h levels).

After some algebraic manipulations, we can notice that the conductance of the horizontal cut is $\varphi(G')_{S_h} = \frac{1}{h-1}$. Similarly, the conductance of vertical cut is:

$$\varphi(G')_{S_v} = \begin{cases} \frac{h}{nd(h-1)} & \text{if } d \le \frac{n}{2h} \\ min\left(\frac{2hd-n}{nd}, \frac{1}{h-1}\right) & \text{if } \frac{n}{2h} < d < \frac{n}{h}. \end{cases} \tag{4}$$

The conductance of the graph is $min(\varphi(G')_{S_h}, \varphi(G')_{S_v})$. In order to analyze the impact of intra level edges, we assume a simple model where each node has k randomly chosen intra-layer edges. We can see that the horizontal cut for this graph: $\phi(G)_{S_h} = 1/(h-1+hk/(2d))$. i.e. the horizontal cut with intra level edges decrease the conductance. There are four possible cases for vertical cut depending on the value of d and h. The second argument for min in Equation 2 provides the value for $\phi(G)_{S_v}$. Comparing the equations, we can notice that the additional factor of k (introduced due to intra level edges) actually reduces conductance. □

One can see from the theorem, specifically the comparison between (2) and (3) that the removal of intra-level edges significantly increases the graph conductance and thereby make the random walk process more efficient. Our experiments on the Twitter prototype verified this finding. Figure 4 shows, for various keywords , how the removal of 10% to 100% randomly chosen intra-level edges affect the query cost of simple random walks to achieve a relative error of $\le 5\%$ on estimating the average number of followers for all users who tweeted the keyword in 2013. One can observe from the figure that as the query cost decreases dramatically when intra level edges are removed. Even removal of subset of such edges is actually helpful. We observed that this modification, on average, reduces the query cost for most keywords by at least 20%.

4.2.3 Time Interval in Level-by-Level Subgraph

We now address the second issue - how to properly set the time interval T which directly affects edge classification. Once again, we start with theoretical analysis on optimal T based on the simple example of level-by-level graph described in §4.2.2. Then, we verify the analysis with experimental findings over Twitter.

Theoretical Analysis: Note that the setting of T affects two parameters in this simple model: (1) the number of levels h - the longer T is, the smaller h, and (2) d, the number of (randomly chosen) Level $i+1$ nodes a Level i node is connected with. Here the relationship between T and d is not as clear: While a longer T will in general lead to more nodes on Level $i+1$, it *might* actually reduce d if most followers of the Level i node already responded within the time interval corresponding to Level i. The following corollary to Theorem 4.1 illustrates the relationship between h and d in order to maximize conductance of the level-by-level subgraph.

COROLLARY 4.1. *To maximize the conductance of G', there is*

$$d = \frac{(2h-1)(2h-2)}{h(2h-9)} \tag{5}$$

The proof follows directly from Theorem 4.1. Intuitively, this means that instead of setting the T to a fixed value, we should adjust it according to the propagation pattern of the query term or hashtag. Specifically, the average number of followers who "pick up" the hashtag after the current time interval should be close to its optimal value d as shown in (5). For example, if the average degree is around $d = 14$, then there should be around $h \approx 5$ levels in the lattice structure. Of course, the real-world scenario is more complex. For example, the average number of "pick ups" tends to decline over time - indicating that the time interval should be dynamically changed throughout the duration of propagation [3, 18].

Another interesting observation from the corollary is that the optimal value of d becomes very close to 2 (i.e., its limit when $h \to \infty$) when h is reasonably large. For example, we have $d = 2.13$ and 2.06 when $h = 50$ and 100, respectively. This means that when the keyword of interest has been propagated for a long time (e.g., `privacy`), we can set T according to a simple rule of $d = 2$.

Practical Design: Recall that GRAPH-BUILDER constructs the best subgraph *on-the-fly* during aggregate estimation. First, we discuss a simpler problem where we are given a set of candidate values for T and aim is to identify which is best for estimating an aggregate. Constructing the term-induced subgraph for each value and comparing them is not ideal as it would require a prohibitive query cost. Instead, we perform a *pilot* random walk using each of the time intervals. Each of the pilot random walks uses a smaller budget (e.g., 50 samples) and terminates quickly. Using the partial topology revealed by each walk, we compute h and d and estimate the value of conductance using (3). The time interval with the highest conductance is selected and used for the rest of the process.

We evaluated the effectiveness of this over Twitter. Specifically, we identified a set of diverse time intervals varying from 1-hour to 1-month. For each time interval, we estimated its efficacy in sampling as against the theoretical value of the conductance. In other words, we ordered the time intervals in the horizontal x axis based on their conductance. We then performed random walk for each of these time intervals and compared the query cost to achieve a relative error of less than 5%. Figure 5 shows the results for three keywords. One can see that the orders based on theoretical conductance and experimental performance are consistent.

Algorithm MA-SRW: Recall from §3 the two key components of MICROBLOG-ANALYZER: GRAPH-BUILDER and GRAPH-WALKER. In this section, we developed a level-by-level subgraph for GRAPH-BUILDER. We now combine it with simple random walk in GRAPH-BUILDER to produce Algorithm MICROBLOG-ANALYZER-Simple Random Walker (MA-SRW). The samples obtained are then used for aggregate estimation in the same way as simple random walk [20]. Algorithm 1 depicts the pseudocode for MA-SRW.

Algorithm 1 MA-SRW

```
1: Retrieve seed users
2: while Remaining query budget $> 0$ do
3:     Retrieve samples using simple random walk. At each transition, only select from edges that belong to the level-by-level graph according to time interval $T$.
4: end while
5: Perform aggregate estimation as in simple random walk [20].
```

5. TOPOLOGY-AWARE RANDOM WALKS

To understand the key ideas of our Topology-Aware random walk algorithm, we start by briefly discussing the deficiencies of existing techniques, specifically the direct application of simple random walk or Metropolis-Hastings random walk to the level-by-level subgraph we constructed in Section 4. Then, we develop the key ideas for a novel topology-aware, level-by-level, random walk and present our MA-TARW algorithm.

5.1 Deficiencies of Traditional Random Walks

As mentioned in the introduction, the existing techniques have two main problems: (1) although they produce asymptotically unbiased (according to their respective stationary distributions) samples after a burn-in period, the number of transitions required for the burn-in is usually high [25]; and (2) while they can be combined with mark-and-recapture [9] to estimate SUM and COUNT queries based on the samples, the query cost often rises to a prohibitively high level for practical purposes.

We note here that the fundamental reason underlying these problems is the *inability* of traditional random walk techniques to estimate the probability for a node u to be chosen as a sample. Note that while simple random walk is known to have a stationary distribution that assigns probability proportional to a node's degree $d(u)$, it is still impossible to compute the exact probability for a node to be accessed (i.e., $d(u)/(2|E|)$ where E is the set of all edges) unless one knows the total number of edges in the graph. Similarly, to know the exact probability for a node to be accessed by Metropolis-Hastings random walk (i.e., $1/|V|$ where V is the set of all vertices), one has to know the total number of nodes in the graph. Clearly, neither piece of knowledge is available *a priori* in our case - and estimating them (e.g., by using mark-and-recapture) requires a very high query cost.

To understand the importance of knowing the exact probability for a node to be accessed, note that such knowledge indeed addresses both deficiencies outlined above. First, with knowledge of $p(u)$, the probability for a node to be taken as a sample, one can simply apply the Hansen-Hurwitz estimator [14] to generate an *unbiased* estimation for any SUM or COUNT query defined in §2 as $f(u)/p(u)$, where $f(u)$ is the result of applying the SUM or COUNT query over u itself[3]. This avoids the usage of mark-and-recapture and, as a result, significantly reduces the query cost required for answering SUM and COUNT queries[4].

Similarly, the efficiency problem - i.e., the long burn-in period required - is also (at least partially) caused by the lack of knowledge on the probability for a node to be sampled at a certain step of the random walk. Specifically, the lack of knowledge mandates a long burn-in period for the sampling probability to converge to its target stationary distribution. If one can compute $p(u)$ during each step of the random walk process, then an unbiased aggregate estimation can be generated as long as $p(u) > 0$ for all u in the graph[5] - potentially saving significant query cost for the sampling process.

Admittedly, if one has no knowledge of the global graph topology, it is impossible to compute or make any meaningful estimation[6] of $p(u)$ without incurring as high a query cost as mark-and-recapture [9, 15]. The reason is simple - without "recapturing" at least some nodes accessed before, it is impossible to determine the scale of the graph as, theoretically speaking, it is entirely possible that the access cost we have incurred so far is still smaller than the average pairwise distance between nodes in the graph (one can always construct such an extreme-case scenario), making it impossible to guarantee or even estimate the error of aggregate estimations.

Fortunately, the subgraph construction technique described in §4 affords us substantial knowledge of the graph topology - not the entire node/edge sets - but knowledge of the level-by-level structure all nodes and edges are organized by, and which level a node falls into. As we shall show in the following subsection, such knowledge gives us the ability to efficiently compute an unbiased estimation of $p(u)$, which in turns enables a significantly more efficient (topology-aware) sampling process than the traditional random walk techniques.

5.2 Key Idea: Level-by-Level Random Walk

In this section, we first develop a novel level-by-level random walk process by leveraging knowledge of the subgraph topology we constructed in §4. We also explain why this process requires far fewer queries than traditional (simple or Metropolis-Hastings) random walks. Then, we discuss how to estimate $p(u)$ in a level-by-level random walk - which in turn enables accurate aggregate estimations.

Description of Level-by-Level Random Walk: To understand the level-by-level random walk process, we start by considering a simple example where a level-by-level subgraph constructed for a given keyword has h levels and only edges between nodes of adjacent levels. As shown in Figure 6, the top level consists of users who mentioned the keyword earliest, while users at the bottom one or few levels are guaranteed to be returned by Twitter's search API (which has a time limit of about 1 week [6]) - i.e., our random walk process starts from these bottom levels. Note that every edge in the graph is directed from top to bottom.

Our topology-aware, level-by-level, random walk follows a bottom-top-bottom flow on the subgraph - i.e., a *random walk instance* starts from the bottom level and moves up one level at a time, by following the inverse direction of edges, until reaching a node with no incoming edge. Then, it reverses traversal direction and starts

[3]e.g., if the aggregate is the number of posts containing `privacy`, then $f(u)$ is the number of u's posts containing `privacy`.

[4]Note that AVG queries can be simply estimated as SUM/COUNT.

[5]Note that the requirement $p(u) > 0$ is here to ensure that the sampling process can reach all nodes covered by the aggregate.

[6]Here we use "meaningful" to refer to estimations with statistical guarantees on bias and/or variance.

following the original edge directions to transit down, again one level at a time, until it reaches a node with no outgoing edge - at which time (this instance of) the random walk terminates. At each transition during the random walk, a branch is chosen uniformly at random. Note that *all nodes* we pass through during a random walk will be used to generate one aggregate estimation - and one can execute multiple instances of the random walk and average out the results to produce more accurate estimations - the details of these issues shall be described in the next subsection.

Before discussing the probability for each node to be chosen by such a level-by-level random walk, we first note that the query cost required by each instance of the random walk is much smaller than that for traditional topology-oblivious random walks. Specifically, our walk instance requires at most $2(h-1)$ transitions, orders of magnitude fewer than simple and Metropolis-Hastings random walks, according to the results in §6.2.

There is a simple reason behind this advantage: by leveraging knowledge of the level-by-level topology, our random walk process is capable of transiting between different "clusters" of nodes much faster than traditional topology-oblivious random walks. Specifically, for a $2(h-1)$-step level-by-level random walk instance over the above-described h-level graph, each of the first (or last) $h-1$ steps is guaranteed to draw from mutually exclusive subsets of nodes. This makes the random walk process reach (with a positive probability) all nodes in the graph much faster than traditional random walks which, despite improved subgraph designs, still have a fairly high probability to return to their origin point after a small number of transition steps [10].

Unbiased Estimation of $p(u)$: We now consider the estimation of $p(u)$ - the probability for a level-by-level random walk instance to reach a node u in the subgraph. To do so, we first define some notation. We use $\acute{p}(u)$ and $\grave{p}(u)$ to represent the probability for a random walk to reach u during the bottom-top and top-bottom phases, respectively. Also, we use $\nabla(u)$ and $\Delta(u)$ to denote the set of neighbors of u on the levels above and below u, respectively. The key observation for estimating $\acute{p}(u)$ and $\grave{p}(u)$ is

$$\acute{p}(u) = \sum_{v \in \Delta(u)} \frac{\acute{p}(v)}{|\nabla(v)|}, \qquad \grave{p}(u) = \sum_{v \in \nabla(u)} \frac{\grave{p}(v)}{|\Delta(v)|}, \tag{6}$$

which holds for all but two exceptions: (1) for a node u with no incoming edges - i.e., when $\nabla(u) = \varnothing$ - we have $\grave{p}(u) = \acute{p}(u)$, and (2) for a node u with no outgoing edges - i.e., when $\Delta(u) = \varnothing$ - it is either $\acute{p}(u) = 1/s$ - where s is the number of seed nodes[7] the random walk might start from - if u is one of the seed nodes, or $\acute{p}(u) = 0$ otherwise.

Equation 6 illustrates a simple recursive process for producing an unbiased estimation of $p(u)$: Note that if we choose a node v uniformly at random from $\Delta(u)$, then

$$\omega(\acute{p}(u)) = \frac{|\Delta(u)| \cdot \acute{p}(v)}{|\nabla(v)|} \tag{7}$$

is an unbiased estimation for $\acute{p}(u)$ (same[8] applies to $\grave{p}(u)$). In addition, if we replace $\acute{p}(v)$ in (7) with an unbiased estimation of it, say $\omega(\acute{p}(v))$, then $|\Delta(U)| \cdot \omega(\acute{p}(v))/|\nabla(v)|$ remains an unbiased estimation of $\acute{p}(u)$ as long as the random selection of v from $\nabla(u)$ is independent of the estimation of $\omega(\acute{p}(v))$.

Algorithm 2 ESTIMATE-$\acute{p}$

```
1: Input: $u$
2: if $\Delta(u) == \varnothing$ then
3:     //$u$ is a bottom level node
4:     $\acute{p}(u) = \begin{cases} 1/s & \text{If } u \text{ is one of the } s \text{ seeds} \\ 0 & \text{Otherwise} \end{cases}$
5: else if $\nabla(u) == \varnothing$ then
6:     //$u$ is a top level node
7:     $\acute{p}(u) = \grave{p}(u)$
8: else
9:     Pick a node $v$ randomly from $\Delta(u)$
10:    $\acute{p}(v)$ = ESTIMATE-$\acute{p}$ $(v)$
11:    $\acute{p}(u) = \frac{|\Delta(U)| \cdot \acute{p}(v)}{|\nabla(v)|}$
12: end if
```

As such, the recursive process works as follows: After each instance of the level-by-level random walk terminates, we take $\acute{U}$ and $\grave{U}$, the sets of nodes the instance passes through during the bottom-top and top-bottom phases, respectively. Then, for each node $u \in \grave{U}$, we start a bottom-top, level-by-level random walk starting from u, this time for the sole purpose of recursively estimating $\grave{p}(u)$. On the other hand, for each node $u \in \acute{U}$, we start a top-bottom level-by-level random walk to estimate $\acute{p}(u)$ in a recursive fashion. Algorithm 2 depicts the pseudocode for estimating $\acute{p}(u)$ (the algorithm for $\grave{p}(u)$ is similar). One can see that this process can produce unbiased estimations of $\grave{p}(u)$ or $\acute{p}(u)$ for every node that the random walk instance passes through - i.e., every node that will be used in the aggregate estimation process, as explained in the next subsection.

Since the above discussions have established the unbiasedness of $f(u)/\grave{p}(u)$ on SUM and COUNT estimations as well as the unbiasedness of $\omega(\grave{p}(u))$ on estimating $\grave{p}(u)$, we now consider the other important factor affecting the error of aggregate estimation: *variance*. Specifically, the following theorem illustrates the estimation variance produced by topology aware random walk for SUM aggregates. Note that since COUNT can be considered as a special case of SUM (when $f(u) = 1$), estimation errors of COUNT and AVG (i.e., SUM/COUNT) aggregates can be derived accordingly.

THEOREM 5.1. *For aggregate Q_{A}:* `SELECT SUM(`$f(u)$`) FROM U WHERE` *cond, after r random walk instances, topology aware random walk generates an estimation variance*

$$\sigma^2 = \left(\sum_{u \in cond} \frac{(V+1) \cdot f(u)^2}{r \cdot \grave{p}(u)} \right) - \frac{Q_{\mathrm{A}}^2}{r}, \textit{ where} \tag{8}$$

$$V = \sum_{u \in cond} \sum_{\rho \in \mathcal{P}(u)} \grave{p}(u) \cdot p(\rho) \cdot \left(\frac{\grave{p}(u)}{\omega(\rho)} - 1 \right)^2 \tag{9}$$

when r is sufficiently large, where Q_{A} is the real aggregate value, $\mathcal{P}(u)$ is the set of all bottom-top-bottom paths from u to one of the seed nodes, $\omega(\rho)$ is the estimation of $\grave{p}(u)$ produced by Algorithm 2 when path ρ is taken for estimating $\grave{p}(u)$, and $p(\rho)$ is the probability for ρ to be taken.

We do not include the proof here due to space limit. Note that an intuitive explanation for V in the theorem is the variance of $\grave{p}(u)/\omega(\grave{p}(u))$, where $\omega(\grave{p}(u))$ is the estimation of $\grave{p}(u)$ produced by our algorithm, taken over the randomness of $\omega(\grave{p}(u))$. One can observe from the theorem that a key factor determining the estimation variance is the values of $\grave{p}(u)$ for nodes in the subgraph. To understand why, note from (8) that, given V, σ^2 is in general inversely proportional to $\grave{p}(u)$. Thus, if the subgraph happens to be

[7]Recall from §3 that seed nodes consist of users returned by the limited search interface - e.g., for Twitter, those who tweeted the keyword within the last week and thus returned by the Search API.

[8]i.e., if we choose a node v uniformly at random from $\nabla(u)$, then $|\nabla(u)| \cdot \grave{p}(v)/|\Delta(v)|$ is an unbiased estimation for $\grave{p}(u)$.

highly skewed so as to have a node u with an extremely small $\grave{p}(u)$, then the estimation variance σ^2 (and thereby the aggregate estimation error) can be very large. Fortunately, as we shall show in §6, the variance is indeed fairly small in practice for the wide variety of keywords we tested.

Before concluding this subsection, we would like to briefly discuss the additional query cost introduced by the probability estimation process. One can see that, in order to estimate $\grave{p}(\cdot)$ or $\acute{p}(\cdot)$ for the (at most) $2h - 1$ nodes the random walk passes through, this process requires at most $(2h - 1) \cdot (h - 1)$ additional transitions. While such $O(h^2)$ query cost surpasses that required by the level-by-level random walk itself, it is unlikely to cause any efficiency concern in practice because of the following two reasons.

First, as one can see from the results in §6.2, even a query cost of $O(h^2)$ is still an order of magnitude lower than topology-oblivious random walks, and second, the real-world query cost for estimating $\grave{p}(\cdot)$ or $\acute{p}(\cdot)$ is often lower than the worst-case scenario. To understand why, consider a common scenario where the level-by-level subgraph has one or a small number of roots at the top. Let there be one root v_{r}. Note that once we produce an estimation of $\grave{p}(v_{\mathrm{r}})$ (which is equal to $\acute{p}(v_{\mathrm{r}})$), we can reuse it for estimating $\grave{p}(\cdot)$ for *all* nodes in the top-bottom phase of *all* random walk instances - i.e., for these nodes, the probability estimation process only needs to walk bottom-up and not top-bottom anymore - saving about half of the query cost because of a single cache.

5.3 Algorithm MA-TARW

In this subsection, we put together the previous discussions of level-by-level subgraph, topology aware random walk and the unbiased estimation of selection probability $\grave{p}(u)$ to develop Algorithm MA-TARW, which can be used to estimate SUM, COUNT and AVG aggregates with or without selection conditions.

Algorithm 3 depicts the pseudocode for MA-TARW. First, it uses a small number of bootstrapping transitions to identify the best time interval T for the level-by-level subgraph (see §4.2.3 for details). It randomly picks a bottom level node (a user who has recently tweeted about the hashtag) and performs a bottom-top-bottom random walk instance $\mathcal{R}_i$ as described in previous subsection. For each node u in the walk $\mathcal{R}_i$, it computes the selection probability ($\acute{p}(u)$ or $\grave{p}(u)$). All nodes in $\mathcal{R}_i$ are used in computing a single estimate of the aggregate query. This random walk process is then repeated for multiple times - with the average estimate being outputted as the final aggregate estimation.

Algorithm 3 MA-TARW

```
1: Estimate best value of $T$ using bootstrapping transitions
2: while Remaining query budget $> 0$ do
3:     Perform a bottom-top-bottom random walk $\mathcal{R}_i$
4:     $\acute{p}(u)$ = ESTIMATE-$\acute{p}$ $(u)$ $\forall\, u \in \acute{U}$ of $\mathcal{R}_i$
5:     $\grave{p}(u)$ = ESTIMATE-$\grave{p}$ $(u)$ $\forall\, u \in \grave{U}$ of $\mathcal{R}_i$
6:     // Remove nodes from $\acute{U}$, $\grave{U}$ that does not match input query
7:     $\widetilde{f}(\mathcal{R}_i) = \frac{1}{|\mathcal{R}_i|}\left(\sum_{u\in\acute{U}} \frac{f(u)}{\acute{p}(u)} + \sum_{u\in\grave{U}} \frac{f(u)}{\grave{p}(u)}\right)$
8: end while
9: Return average of all previous estimates $\widetilde{f}(\mathcal{R}_i)$
```

6. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Hardware and Platform: All our experiments were conducted on a computer with Intel Core(TM) i5 2.50 GHz CPU with 8 GB of RAM. The algorithms were implemented in Python 2.7.

Datasets: We tested our algorithms on three real-world microblogging platforms - Twitter, Google+ and Tumblr. These were chosen due to their popularity and accessibility of their developer API. While the majority of our experiments were conducted over Twitter, we observed similar behavior on the other microblogs also.

Detailed discussion on how MICROBLOG-ANALYZER is instantiated for Twitter is found in §3.2. We now briefly describe how Google+ and Tumblr are instantiated. Google+ has an *Activity* API (equivalent to the Twitter search API) that allows us to search for posts that specify a particular keyword. It also has an API to retrieve user profile information, as allowed by the privacy setting of the user. However, some basic information such as display name are always available. Similar to Twitter, connections in Google+ are asymmetric. Connections are grouped into various groups, called *Circles*. Google+ has a *courtesy* rate limit of 10,000 queries per day and 5 per second. Due to the difficulty in retrieving connections (the API only provides the connections of an authenticated user), we define two users to be connected if they performed some activity together in last year, i.e., they liked, shared or commented the same post. Tumblr is another popular microblogging platform where users host multiple blogs and can follow blogs of other users. The posts in blogs correspond to tweets in Twitter which can then be liked or reblogged by other users. Tumblr has extensive API to retrieve various information about blogs. Requests are rate-limited to one every 10 seconds.

Aggregate Queries and Ground Truth: In our experiments, we focused on aggregate queries AVG, COUNT and SUM. We evaluated aggregate measures such as the number of followers, display name length, number of likes in the blog etc. For Twitter, we used the streaming API to collect all public tweets mentioning a diverse set of keywords (such as cities, celebrities, organizations etc) between Jan 1 - Nov 1, 2013. Since Twitter ensures that the stream returns all relevant tweets as long as their frequency is less than 1% of the Twitter Firehose and our keywords are not too frequent, this provides a reasonable ground truth over which aggregate estimation algorithms can be evaluated. Figure 7 shows the frequency of three keywords used in the evaluation over time - `privacy` (a relatively low frequency term with occasional spikes), `New York` (a perpetually popular and high frequency keyword) and `Boston` (keyword that has medium frequency but had a singular spike on Apr 15, 2013 when the Marathon Bombing occurred). For Google+ and Tumblr, no convenient way to collect ground truth exists. To get a reasonable approximation, we instantiated multiple samplers that performed simultaneous random walks until they converged to their stationary distribution (with Geweke threshold $Z \leq 0.05$). The average estimate from all the walks serves as ground truth.

Performance Measures: Our aggregate estimation algorithms were evaluated according to two measures. Efficiency was measured as the number of API interface calls. Notice that multiple API calls could be required to obtain the result of a single query. For example, Twitter's followers API returns 5000 users per call and hence multiple calls are required to retrieve all followers of a celebrity. To measure accuracy, we use the relative error (see §2).

Algorithms Evaluated: We evaluated MA-SRW, MA-TARW and the state-of-art baseline M&R described next. Recall from §4 that MA-SRW outperformed SRW on the original or the term-induced social graph. Hence, to keep the presentation clear, we do not present any experiments on the original or the term-induced social graph. To the best our knowledge, we have not found any research that performs general aggregate estimation over microblogs. The closest is [15] that performs size (COUNT) estimation for (entire) social networks and does not directly support keyword-specific size estimation. We adapted [15] to only consider nodes that match the

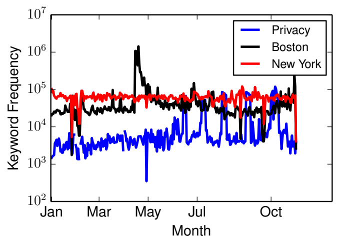

Figure 7: Frequencies of Chosen Keywords

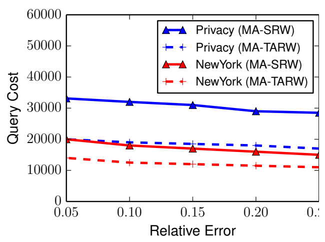

Figure 8: Twitter: AVG(followers)

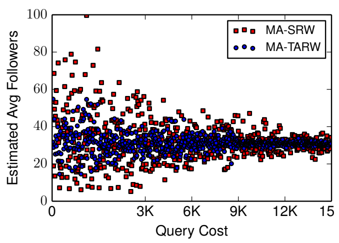

Figure 9: Twitter: Estimated AVG(followers)

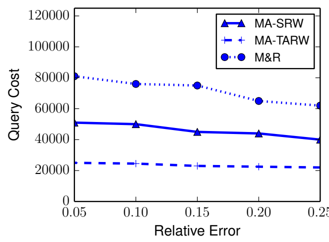

Figure 10: Twitter: Count(users)

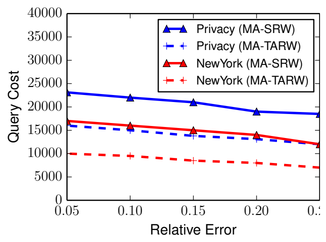

Figure 11: Twitter: AVG(Display Name)

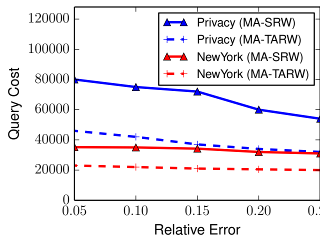

Figure 12: Google+: AVG(Display Name)

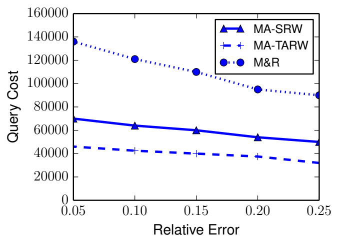

Figure 13: Google+: COUNT (male users who tweeted)

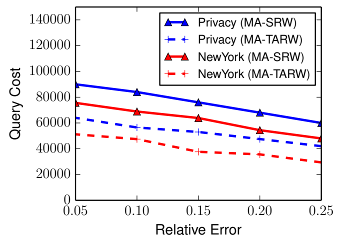

Figure 14: Tumblr: AVG(Likes)

Table 3: Average Percent Improvement of MA-TARW

KEYWORD	MA-SRW (AVG)	MA-SRW (COUNT)	M&R (COUNT)
`Boston`	39	44	72
`Oprah`	27	37	67
`Simvastatin`	29	41	74
`$WMT`	33	51	78
`Lipitor`	24	47	76
`Tunisia`	33	31	53
`Tahrir`	41	55	61

query and used it to measure the size of the term induced subgraph and refer this algorithm as M&R (for mark and recapture); we only include it for COUNT (as the algorithm was designed for).

6.2 Experimental Results

We start with Twitter. Table 3 shows the average percentage improvement in Twitter query cost achieved by MA-TARW over MA-SRW and M&R for AVG(followers) and COUNT(users) queries (from Jan 1, 2013 to Oct 31, 2013) involving diverse keyword conditions to achieve a relative accuracy error of 5%. The results show that MA-TARW outperforms both competing algorithms and confirm our theoretical analysis.

Next, we study in more detail specific aggregate queries. We use MA-TARW to estimate the average number of followers of all users who tweeted `privacy`. Figure 8 shows that MA-TARW significantly outperforms MA-SRW. Figure 9 also validates this conclusion by showing that MA-TARW converges to the true estimate and has a lower variance in its estimate within few thousand queries.

We then perform a COUNT estimate of all users who tweeted `privacy`. Figure 10 shows that MA-TARW outperforms both MA-SRW and baseline M&R. Recall from Figure 3 that M&R requires lower query cost when evaluated on the level-by-level subgraph than on term induced subgraph; this is why we execute M&R on the level-by-level subgraph to better evaluate our topology-aware navigation algorithm. We next consider an aggregate query to estimate the average display name length of Twitter users who tweeted `privacy`. In contrast to AVG(#followers) shown above, this requires substantially smaller number of queries as this measure has a lower variability than that of number of followers. Figure 11 shows that MA-TARW seems to leverage this aspect by essentially "skipping" such edges (which would have often been intra-level edges).

Next we evaluate our algorithms on Google+. Figures 12 and 13 show the performance of estimating the average display name length and count of male users (gender is generally missing from Twitter profiles, and hence we did not use it as a condition above) who posted `privacy` during the time period. We notice that MA-TARW outperforms the competing algorithms. It must be noted that the absolute query cost is much higher than in Twitter. This is to a large extent due to the fact that APIs of Google+ (such as Activity search) returns at most 20 results per invocation compared to 200 in Twitter's timeline API.

Finally, we evaluate our algorithms on Tumblr. Here, we evaluated the average number of likes obtained by posts with *textual* content containing the keyword `privacy`. Figure 14 shows that MA-TARW has the best performance.

7. RELATED WORK

Graph Sampling Through Random Walks: A number of existing papers have studied the problem on sampling large graphs [8, 17, 19, 20] while [13, 15, 25] specifically focus on online social networks. Sampling techniques and the ground truth definition vary depending on whether the global topology is known [13, 19] or unknown. For the latter, [13, 19] compared the efficiency of various sampling techniques such as simple random walk (SRW), Metropolis-Hastings (MHRW), BFS and DFS. [13] also studied the problem of running multiple, parallel random walks. We used SRW as the basis of MA-SRW as [13] reported that SRW is typically 1.5-8 times faster than MHRW, which was observation as well.

Analytics of Twitter and Other Microblogs: While there has been plethora of work on using social media data from Twitter and other microblogs on specific analytics tasks (typically over current and future data), our paper is the first to study the problem of aggregate estimation over historic data. [16] provides an high level overview of possible analytics tasks over Twitter. Other analytics tasks include monitoring trends [22], predicting stock prices [28], topical expertise [27], measure information propagation in Twitter [23], such as in the context of natural disasters. There has been a set of paid and free third party services such as Sysomos, Topsy, Trendsmap etc that allow you to perform simple analytics tasks (such as monitor popular trends, analyze your tweeting/retweeting behavior, visualize your social network etc). However, none of the free ones allow analytics over historic data and even the paid ones offer simple, canned analytic options.

Search Engine Analytics: Another category of related research is analytics over a search engine's corpus (e.g., [29]) - simply because a microblog service can be considered as a search engine (collecting, indexing and publishing documents posted by all users). However, search-engine-analytics techniques cannot be directly applied because of the limitation of search interface provided by microblogging services. Note that a key assumption made by all existing search-engine-analytics techniques is that the search interface can reveal *all* documents in the corpus (through answers to a very large set of search queries). This, unfortunately, is not the case for microblogging services. For example, Twitter search API is limited to tweets published in the last week [6]. These limitations prevent existing search-engine-analytics techniques from being applied.

8. CONCLUSIONS AND FUTURE WORK

We proposed novel solutions to perform aggregate query estimation on microblogging data that exploit the provided user timeline API calls. We showed how to define a conceptual level-by-level subgraph of the social graph that allows dramatically more efficient random walk-based sampling. Then, we further improved our solution by proposing a novel topology-aware navigation strategy on the level-by-level subgraph that significantly outperforms existing random walk sampling methods. Theoretical analysis and experiments over microblogs confirm the effectiveness of our solutions.

9. ACKNOWLEDGEMENTS

The work of Saravanan Thirumuruganathan and Gautam Das was partially supported by National Science Foundation under grants 0812601, 0915834, 1018865 and grants from Microsoft Research. Nan Zhang was supported in part by the National Science Foundation under grants 0852674, 0915834, 1117297, and 1343976. Vagelis Hristidis was partially supported by National Science Foundation grant 1216007 and a Samsung GRO grant. Any opinions, findings, conclusions, and/or recommendations expressed in this material, either expressed or implied, are those of the authors and do not necessarily reflect the views of the sponsors listed above.

10. REFERENCES

[1] Datasift pricing. In *http://datasift.com/pricing/*, 2013.

[2] Gnip. In *http://gnip.com*, 2013.

[3] Sysomos twitter retweet statistics. In *http://www.sysomos.com/insidetwitter/engagement/*, 2013.

[4] Sysomos twitter usage statistics. In *http://www.sysomos.com/insidetwitter/*, 2013.

[5] Twitter api. In *https://dev.twitter.com/docs/api/1.1*, 2013.

[6] Twitter search. In *https://dev.twitter.com/docs/using-search*, 2013.

[7] Twitter Streaming API. In *https://dev.twitter.com/docs/streaming-apis*, 2013.

[8] E. M. Airoldi and K. M. Carley. Sampling algorithms for pure network topologies: a study on the stability and the separability of metric embeddings. *ACM SIGKDD Explorations Newsletter*, 7(2):13–22, 2005.

[9] L. Cowen. Handbook of Capture-Recapture Analysis. *The Quarterly Review of Biology*, (3):310.

[10] C. Domb. On multiple returns in the random-walk problem. In *Proc. Cambridge Philos. Soc*, volume 50, pages 586–591. Cambridge Univ Press, 1954.

[11] J. Geweke et al. *Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments*. Federal Reserve Bank of Minneapolis, Research Department, 1991.

[12] W. R. Gilks. *Markov Chain Monte Carlo In Practice*. Chapman and Hall/CRC, 1999.

[13] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: a case study of unbiased sampling of osns. INFOCOM'10, pages 2498–2506, 2010.

[14] M. H. Hansen and W. N. Hurwitz. On the theory of sampling from finite populations. *AMS*, 14(4):333–362, 1943.

[15] L. Katzir, E. Liberty, and O. Somekh. Estimating sizes of social networks via biased sampling. WWW '11, 2011.

[16] S. Kumar, F. Morstatter, and H. Liu. Twitter data analytics, 2013.

[17] M. Kurant, M. Gjoka, C. T. Butts, and A. Markopoulou. Walking on a graph with a magnifying glass: stratified sampling via weighted random walks. SIGMETRICS '11.

[18] K. Lerman and R. Ghosh. Information contagion: An empirical study of the spread of news on digg and twitter social networks. *ICWSM*, 10:90–97, 2010.

[19] J. Leskovec and C. Faloutsos. Sampling from large graphs. KDD '06, pages 631–636, 2006.

[20] L. Lovász. Random walks on graphs: A survey. In *Combinatorics, Paul Erdős is Eighty*, volume 2. 1996.

[21] L. Lovasz and R. Kannan. Faster mixing via average conductance. In *STOC*, pages 282–287, 1999.

[22] M. Mathioudakis and N. Koudas. Twittermonitor: Trend detection over the twitter stream. SIGMOD '10.

[23] M. Mendoza, B. Poblete, and C. Castillo. Twitter under crisis: Can we trust what we rt? In *SOMA*, 2010.

[24] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. SIGCOMM '07, pages 29–42, 2007.

[25] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *SIGCOMM '10*, 2010.

[26] M. E. J. Newman. Modularity and community structure in networks. *PNAS*, 103(23), 2006.

[27] A. Pal and S. Counts. Identifying topical authorities in microblogs. In *WSDM*, pages 45–54. ACM, 2011.

[28] E. J. Ruiz, V. Hristidis, C. Castillo, A. Gionis, and A. Jaimes. Correlating financial time series with micro-blogging activity. In *WSDM*, pages 513–522. ACM, 2012.

[29] M. Zhang, N. Zhang, and G. Das. Mining a search engine's corpus: Efficient yet unbiased sampling and aggregate estimation. SIGMOD '11.