Tracking the Conductance of Rapidly Evolving Topic-Subgraphs

Sainyam Galhotra
XRCI, Bangalore
sainyamgalhotra@gmail.com

Amitabha Bagchi
IIT Delhi
bagchi@cse.iitd.ac.in

Srikanta Bedathur
IBM Research
sbedathur@in.ibm.com

Maya Ramanath
IIT Delhi
ramanath@cse.iitd.ac.in

Vidit Jain
American Express Big Data Labs, India
viditumass@gmail.com

ABSTRACT

Monitoring the formation and evolution of communities in large online social networks such as Twitter is an important problem that has generated considerable interest in both industry and academia. Fundamentally, the problem can be cast as studying evolving subgraphs (each subgraph corresponding to a topical community) on an underlying social graph – with users as nodes and the connection between them as edges. A key metric of interest in this setting is tracking the changes to the *conductance* of subgraphs induced by edge activations. This metric quantifies how well or poorly connected a subgraph is to the rest of the graph relative to its internal connections. Conductance has been demonstrated to be of great use in many applications, such as identifying bursty topics, tracking the spread of rumors, and so on. However, tracking this simple metric presents a considerable scalability challenge – the underlying social network is large, the number of communities that are active at any moment is large, the rate at which these communities evolve is high, and moreover, we need to track conductance in real-time. We address these challenges in this paper.

We propose an in-memory approximation called *BloomGraphs* to store and update these (possibly overlapping) evolving subgraphs. As the name suggests, we use Bloom filters to represent an approximation of the underlying graph. This representation is compact and computationally efficient to maintain in the presence of updates. This is especially important when we need to *simultaneously* maintain thousands of evolving subgraphs. BloomGraphs are used in computing and tracking conductance of these subgraphs as edge-activations arrive. BloomGraphs have several desirable properties in the context of this application, including a small memory footprint and efficient updateability. We also demonstrate mathematically that the error incurred in computing conductance is one-sided and that in the case of evolving subgraphs the change in approximate conductance has the same sign as the change in exact conductance in most cases. We validate the effectiveness of BloomGraphs through extensive experimentation on large Twitter graphs and other social networks.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Theory, Conductance, Bloom filters

1. INTRODUCTION

An important problem in the study of social networks is tracking the spread of individual memes or *topics*. These topics may be specific hashtags, URLs, words or phrases or media objects. One natural and important approach in this area has been to study the properties of subgraphs induced on the social network by the users who are posting and propagating these topics (see, e.g. [2, 36, 37]) with the idea that the evolution of these "topic-focused subgraphs" contains information that can help shed light on the viral nature of these topics. Apart from the focus on predicting virality, tracking the spread of topics has important applications in social sciences and market analysis. However, tracking and computing the graph properties of rapidly evolving subgraphs is computationally challenging due to the volume and velocity of the data involved. Not only are there a large number of interactions, these interactions happen within a short span of time, on an underlying network that consists of millions of nodes and edges. For example, in August 2013, Twitter disclosed that an average of $5,700$ tweets were generated *every second* (i.e., around 500 million tweets a day) and activity around a television show made this number peak at $143,000$ tweets per second on 3rd August 2013 [26].

In this paper, we focus on how to track the evolution of topic-focused subgraphs in real-time with very small memory footprint for each subgraph being tracked. The basic setting in which we operate is as follows: we are given an underlying social graph where graph nodes correspond to users and the edges represent their social connections (e.g., the follower-followee relationship on Twitter). As this graph typically changes relatively slowly, we assume it to be static. Topic-focused subgraphs are formed and evolve through activation of edges in this graph. In Twitter, edge-activations correspond to the *retweeting* of or *replying* to a tweet or even *tweeting* within topic. A topic-focused subgraph evolves by including more nodes and edges when users tweet (or retweet) on the same topic, thus (re)activating the edges between them. Such

Proceedings of the VLDB Endowment, Vol. 8, No. 13
Copyright 2015 VLDB Endowment 2150-8097/15/09.

subgraphs can evolve very rapidly, sometimes leading to the topic going viral. For instance, the recent infamous example of the hashtag #JustineSacco went viral within *a few hours* of the first tweet [34].

We are specifically interested in the real-time computation of the *conductance* of these topic-focused subgraphs, a metric that quantifies how well connected the subgraph is to the rest of the graph[1]. Efficiently computing the conductance of a subgraph (or a cut) of a graph has many applications, for example in clustering [24, 28]. In the context of networks, especially social networks, it has been widely used to measure the quality of communities detected by community detection algorithms, and has been shown to be closely related to other measures of the clusteredness of communities like the clustering coefficient [18]. Recently conductance is shown to be an important metric in deciding whether or not a topic has gone viral [2] – which forms a key motivation for our work presented here. The rapidly evolving nature of the topic-focused subgraphs, the size of the underlying graph as well as the need for a real-time solution, makes this a challenging problem. Our solution needs to satisfy the following requirements:

1. Handle a high rate of updates
2. Have low memory footprint for each subgraph being tracked since a large number of subgraphs (i.e., topics) may need to be tracked simultaneously
3. The computation of conductance of the tracked subgraph should be very efficient.

1.1 Contribution

In this paper, we propose the *BloomGraphs* framework which consists of an in-memory approximation of subgraphs that are being tracked, as well as a persistent approximation of the entire underlying social network. As their name suggests, BloomGraphs use Bloom filters to compactly store the adjacency structure of the graph, and use this representation to manage a large number of topic-based subgraphs simultaneously in-memory. Starting with this basic idea, we make the following main contributions:

- We develop a scalable framework for real-time tracking of a large number of topic-based communities simultaneously in memory.
- We show that BloomGraphs allow strictly one-sided error in estimating the conductance of the evolving subgraphs.
- We provide extensions to support conductance tracking under streaming moving time-window scenario.
- We support theoretical claims with empirical evidence over real world graphs derived from a 4-week crawl of close to 8 million users on Twitter as well as other social networks.
- Finally, we show that BloomGraphs are much faster and more space efficient for computing conductance than traditional graph structures.

The key insight behind our definition of BloomGraphs is that computing a graph metric like conductance does not require the full functionality of traditional space-intensive graph storage structures. In particular *we do not always need retrieve a pointer to the neighbors of a node to compute conductance, we can restrict ourselves to checking if a particular node lies in the neighborhood of another*, even if the set membership operation involved throws up some false positives as it does with Bloom filters. Our results demonstrate that for this particular metric the error inherent in Bloom filters can be managed and a quantifiable approximation can be achieved. We are therefore in a position to leverage the tremendous time and space efficiencies Bloom filters offer, making our work an important step towards building systems where structural properties of a large-number of rapidly evolving subgraphs need to be estimated in real-time.

Organization. The rest of the paper is organized as follows: in Section 2, we provide a brief overview of graph dynamics in Twitter that we focus on and the definition of conductance of a subgraph. The Section 3 describes BloomGraphs, the primary contribution of this paper, and develops a theoretical framework to estimate conductance scores using BloomGraphs both in a snapshot setting as well as a streaming setting of edge activations. A detailed experimental evaluation using Twitter crawls as well as a large social network derived from LiveJournal are given in Section 4. Details of related work are given in Section 5, before concluding remarks and outline of future work in Section 6.

2. OVERVIEW

2.1 Data Model

Let $\mathcal{G}(V, E)$ denote the underlying graph, where V and E denote the vertex set and the edge set respectively. In general, $\mathcal{G}$ can either be a directed or an undirected graph, depending on the application. For example, in the case of Twitter network, $\mathcal{G}$ corresponds to the social network between the users induced through directed, follower-followee relationships. In case of Facebook-like social network, the edges could be undirected corresponding to the symmetric friendship relationship.

Based on this, we model interactions between vertices in the network as a sequence of *edge activations* with each activation consisting of a time-stamp, and an associated set of *activation labels*. In real-world terms, this may correspond to a tweet being sent to all the followers, two friends exchanging some message, etc. The activation labels are typically derived from the content of the message/tweet underlying the edge activation and correspond to a topic description.

2.2 Conductance

The *conductance* of a directed graph, $G = (V, E)$ is an isoperimetric quantity that provides a lower bound on the ratio of the outdegree of any set of vertices to their total degree. Formally, for any $S \subseteq V$, we define

$$\phi_G(S) \triangleq \frac{|\{(u, v) \in E : u \in S, v \in V \setminus S\}|}{|\{(u, v) \in E : u \in S\}|},$$

and we define the conductance of the graph G as

$$\phi_G \triangleq \min_{S \subset V} \phi_G(S).$$

Note that the latter quantity ϕ_G is a property of the whole graph while $\phi_G(S)$ is a property of the subset of vertices S. In the rest of this paper we will use the term conductance to refer to the conductance of a subset. When we need to refer to the conductance of the graph we will use the term *graph conductance* to avoid confusion. We will see shortly that from a computational complexity point of view these two quantities differ greatly.

From the definition we can see that the conductance is a measure of how "expansive" or "close knit" the set of vertices is: higher conductance implies more outward connections, lower conductance implies that this cluster of vertices is more "inward-looking." For

[1] It should be noted that this is not the same as computing the conductance of the entire graph, which is defined as the smallest conductance score among all possible subgraphs of the graph.

a more detailed discussion on the use of conductance as a metric, please refer to Section 5.

We only note here that the quantity that we refer to as conductance is also called the *sparsity* of a cut in the algorithms literature since a set of vertices $S \subseteq V$ can be viewed as a cut that divides the graph into two parts $S, V \backslash S$ (see, eg., [27]). In the case of random walks in graphs, or, more generally, for Markov chains on finite state spaces, the term *"bottleneck ratio of a set of states"* has been used by some authors to denote the probability that a random walk (or Markov chain) defined on a set of states V, moves from a subset of states S to a state in $V \setminus S$ (see, e.g., [29]). In the case of random walks this bottleneck ratio is exactly the same as our definition of conductance. Lovász and Simonovits use the term "local conductance" for a similar quantity [30]. Completely unrelated to our setting is the use of the term conductance for individual edges in the study of random walks in terms of electrical resistance (see, again, [29]). Despite the varied uses of the term conductance and for the different names given to the quantity defined above, we use the term conductance for this quantity in this paper since, as we will see in Section 5, this term has been used widely in the context of community detection in networks.

Computation complexity of conductance. To compute the conductance of a set of vertices S we need to determine which of the edges of these vertices are internal to the set, i.e., given an edge (u, v) where $u \in S$ we need to check if $v \in S$. This can be done in time $O(\bar{d}|S|T(|S|))$ where $\bar{d}$ is the average degree of the vertices of S, and $T(k)$ is the time taken to answer a set membership query on a set with k elements. $T(k)$ is typically constant if the set membership structure is a hash map of some sort. Note that this running time is agnostic to the order in which the edges are considered and so if we consider a streaming model where the vertices come in one at a time we can achieve the same time complexity although in this case $T(k)$ may be $O(\log k)$ since we do not know the size of S a priori.

Delving a little deeper, we see that realising the computation of the function $T(\cdot)$ involves storing the set S in an intelligent way to speed up the computation. When we have to compute the conductance for a large set of graphs in parallel then the problem is compounded since a large number of sets have to be stored. This can lead to a significant storage expense, which will further lead to decreased time efficiency. It is this problem that we set out to address in this paper.

We note that although computing the conductance is a polynomial time problem, computing the *graph* conductance, which involves finding the minimum of the conductance over all subsets of the vertex set is known to be a NP-hard problem (see Section 5 for details of what is known about this in the literature.)

3. BLOOMGRAPHS AND CONDUCTANCE

In this section we describe the basic BloomGraph structure and show how to compute conductance using this structure.

3.1 Definition

Since we deal with directed graphs, each node in the graph has a set of edges incoming to it and a set of edges outgoing from it. We define separate notation for these two sets. Given a directed graph $G = (V, E)$ and a $u \in V$, we denote by $\overleftarrow{\Gamma}(u)$ the set $\{v : (v, u) \in E\}$ i.e. the set of nodes that have an outgoing edge that terminates in u and by $\overrightarrow{\Gamma}(u)$ the set $\{v : (u, v) \in E\}$ i.e. the set of nodes that have an incoming edge that originates in u. We will extend this notation to sets of nodes as well in the natural way i.e. for $U \subseteq V$, $\overleftarrow{\Gamma}(U) = \{v : \exists u \in U, (v, u) \in E\}$ and similarly for $\overrightarrow{\Gamma}(U)$. *Only* when we extend this notation to sets of nodes, for convenience of presentation we will allow $\overleftarrow{\Gamma}(U)$ and $\overrightarrow{\Gamma}(U)$ to be multisets, and their cardinality, therefore, will capture the number of edges incoming and outgoing (respectively) to a set of nodes U.

A number of different Bloom filters will appear in the rest of the paper. We use bold math to denote them e.g. $\mathbf{A}$. The size i.e. number of bits of the Bloom filter will be understood by the context and will not be explicitly denoted. In Section 3.3 we will use counting Bloom filters which we will denote with a hat above the symbol i.e. $\widehat{\mathbf{A}}$ to distinguish them from normal Bloom filters. We will also use the following notation for common Bloom filter operations:

- $\mathsf{member}(\mathbf{B}, v)$: Tests whether the element v is stored in the set represented by Bloom filter $\mathbf{B}$.
- $\mathsf{insert}(\mathbf{B}, v)$: Inserts the element v in the set represented by Bloom filter $\mathbf{B}$.
- $\cap(\mathbf{A}, \mathbf{B})$: This returns a new Bloom filter that contains the intersection of the Bloom filters $\mathbf{A}$ and $\mathbf{B}$.
- $\cup(\mathbf{A}, \mathbf{B})$: This returns a new Bloom filter that contains the union of the Bloom filters $\mathbf{A}$ and $\mathbf{B}$.
- $\mathsf{num}(\mathbf{A})$: This returns the approximate number of elements in the set represented by the Bloom filter.

Finally as notation only for the purpose of analysis and not as an operation, we will denote by *size*$(\mathbf{A})$ the actual size of the set stored in the Bloom filter $\mathbf{A}$. Clearly, *size*$(\mathbf{A}) \leq \mathsf{num}(\mathbf{A})$ due to the existence of false positives.

DEFINITION 3.1. *Given a directed graph $G = (V, E)$ and a positive integer parameter m, the* BloomGraph $\mathcal{B}(G, m)$ *of G is the set of Bloom filters $\{\mathbf{In}_G(u), \mathbf{Out}_G(u) : u \in V\}$, where $\mathbf{In}_G(u)$ is a Bloom filter of size m that stores the set $\overleftarrow{\Gamma}(u)$ and $\mathbf{Out}_G(u)$ is a Bloom filter of size m that stores the set $\overrightarrow{\Gamma}(u)$. We refer to $\mathbf{In}_G(u)$ and $\mathbf{Out}_G(u)$ as the* neighbourhood filters *of u. When the graph G is clear from the context we will drop the subscript and simply write $\mathbf{In}(u)$ and $\mathbf{Out}(u)$. Additionally we also store $\left|\overleftarrow{\Gamma}(u)\right|$ and $\left|\overrightarrow{\Gamma}(u)\right|$ for every $u \in V$.*

A BloomGraph is essentially a way of storing adjacency lists using Bloom filters. Unlike in traditional graph representations it is not possible to extract the neighbors of a node from a BloomGraph, we are only allowed to check if a particular node is a neighbour of another node by querying the relevant neighborhood filter using the $\mathsf{member}(\cdot, \cdot)$ operation. When queried, the neighbourhood filter may return some false positives and hence the neighbourhood stored is in effect a superset of the actual neighbourhood. All the Bloom filters used have the same size i.e. m. This makes it possible to efficiently intersect the different Bloom filters, an operation that will be needed often as we will see. Clearly BloomGraphs cannot represent multiedges between a pair of nodes although they can represent self-loops.

3.2 Computing conductance

We now show how to use BloomGraphs to compute conductance. For ease of understanding of our methods we restate the definition of conductance given in Section 2.2 in terms of the notation introduced above: the conductance of a set of vertices $U \subseteq V$ of a graph G,

$\phi_G(U)$ (or simply $\phi(U)$ when the graph concerned is understood) is:

$$\phi(U) = \frac{\left|\overrightarrow{\Gamma}(U)\right|}{\sum_{u \in U} \left|\overrightarrow{\Gamma}(u)\right|}.$$

Algorithm

We now describe an approximate algorithm to calculate the conductance of a subset U using the BloomGraph of G. The algorithm *BloomConductance* takes the $\mathcal{B}(G, m)$ as input along with the set U. The algorithm is iterative. We go through the nodes of U one at a time. For each node we add the size of its outgoing neighborhood to the numerator, subtracting those edges that become internal since they either go from u to the nodes of U that have already been processed, or come in to u from one of those nodes (see Figure 2). If the Bloom filters used had no false positives then the correctness of this algorithm follows from the fact that any edge (u, v) that is internal to U (i.e. $u, v \in U$) gets included in the numerator erroneously when the first of these nodes is processed but is then removed when the second one is processed. However, since each Bloom filter used has some false positives, the conductance computed by *BloomConductance* is approximate.

```
Algorithm BloomConductance$(G, U)$
    Set $num \leftarrow 0$
    Set $den \leftarrow 0$
    /* Initialise numerator and denominator to 0. */
    Set $\mathbf{U} \leftarrow \emptyset$
    /* Bloom filter contains the nodes which we have considered. */
    For each $u \in U$
    /* Iterate over the nodes in $U$. */
        Set $num \leftarrow num + \left|\overrightarrow{\Gamma}(u)\right| - \mathsf{num}(\cap(\mathbf{U}, \mathbf{Out}(u)))$
                    $- \mathsf{num}(\cap(\mathbf{U}, \mathbf{In}(u)))$
        /* Add outgoing edges of $u$, subtract those that go into $U$. */
        Set $den \leftarrow den + \left|\overrightarrow{\Gamma}(u)\right|$
        $\mathsf{insert}(\mathbf{U}, u)$
        /* Add the outgoing edges. */
    Return $\frac{num}{den}$
```

Figure 1: *BloomConductance* run on a vertex set U.

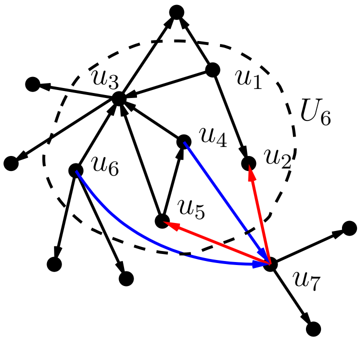


Figure 2: When u_7 is processed the red edges are *not* external since they go into U_6, and the two blue edges which were earlier outgoing from U_6 now become internal.

Theoretical guarantees

We will now see that *BloomConductance* has two important properties. Firstly, as we show in Proposition 3.1 the approximate conductance computed at the end of each iteration is a lower bound on the actual value of the conductance (see (1) below), i.e., the approximate conductance is *always* lower than the exact conductance. We also show that the extent to which the approximate conductance is smaller than the exact conductance can be bounded (see (2) below) although in this case, the result is in terms of the expected value of the approximate conductance. This expectation is over the random choices made in the Bloom filters storing the neighbourhoods of the vertices in the BloomGraph. Overall, Proposition 3.1 shows that the expected value of the conductance computed using the BloomGraph structure is a quantifiable approximation of the exact conductance, and has the additional property that it *never* exceeds the exact conductance, i.e., the error is always one-sided.

We also show that *BloomConductance* has a stronger property that makes it a good approximation even to detect *changes* in the conductance: we state this property as Theorem 3.2: In simple terms what this theorem says is that as long as the nodes of U do not have too high a degree, the change in approximate conductance is in the same direction as that of the exact conductance from step to step i.e. if the conductance increases between step i and step $i + 1$, so does the approximate conductance and if it decreases so does the approximate conductance. We will make the condition for this property to hold precise in the statement of the theorem and discuss the implications of this condition in greater detail in a discussion presented after the proof of the theorem.

In the following we assume that if the algorithm considers the nodes of U in the order $u_1, u_2, \ldots$ then we denote by U_i the set of nodes $\{u_j : 1 \leq j \leq i\}$. We denote the conductance of U_i estimated by *BloomConductance* as $\hat{\phi}(U_i)$, noting that since the Bloom filters used to build the BloomGraph using random hash functions, the quantity $\hat{\phi}(U_i)$ is a random variable. Further, we denote the error in the value of num in the ith iteration of the algorithm by $\Delta(u_i)$ i.e. if $\hat{\alpha}_i = \mathsf{num}(\cap(\mathbf{U}_{i-1}, \mathbf{Out}(u_i))) + \mathsf{num}(\cap(\mathbf{U}_{i-1}, \mathbf{In}(u_i)))$ and $\alpha_i = \left|U_{i-1} \cap \overrightarrow{\Gamma}(u_i)\right| + \left|U_{i-1} \cap \overleftarrow{\Gamma}(u_i)\right|$, then $\Delta(u_i) = \hat{\alpha}_i - \alpha_i$.

Further, we note that the false positive probability for a Bloom filter of size m that has had ℓ items inserted in it using k hash functions was calculated by Broder and Mitzenmacher [8] to be

$$p_{m,k}(\ell) = \left(1 - \left(1 - \frac{1}{m}\right)^{k\ell}\right)^k,$$

and so $\beta_{N,m,k}(\ell) = (N - \ell)p_{m,k}(\ell)$. Where m and k are understood, we will simply write $p(\ell)$ for these two quantities. Now we move to the results.

Proposition 3.1. *The approximate value of conductance $\hat{\phi}(U_i)$ calculated at the end of the ith iteration of the algorithm* BloomConductance(G, U) *has the following properties for all $1 \leq i \leq |U|$:*

$$\hat{\phi}(U_i) \leq \phi(U_i), \qquad (1)$$

$$(1 - p_{m,k}(i-1)(1 + \mathit{flux}(U_i))) \cdot \phi(U_i) - c \leq E[\hat{\phi}(U_i)], \qquad (2)$$

where $p_{m,k}(i)$ is the false positive probability for a Bloom filter of size m with i elements stored in it using k hash functions, the flux, flux(U), of a set of nodes U, is defined as the average incoming degree of U divided by the average outdegree of U, and c is a small constant that depends on the maximum incoming and outgoing degrees of U_i and not on its conductance.

PROOF. The proof of (1) is straighforward:

$$\hat{\phi}(U_i) = \frac{\sum_{j=1}^{i}\left(\left|\overrightarrow{\Gamma}(u_j)\right| - \hat{\alpha}_j\right)}{\sum_{j=1}^{i}\left|\overrightarrow{\Gamma}(u_j)\right|} \leq \frac{\sum_{j=1}^{i}\left(\left|\overrightarrow{\Gamma}(u_j)\right| - \alpha_j\right)}{\sum_{j=1}^{i}\left|\overrightarrow{\Gamma}(u_j)\right|} = \phi(U_i),$$

where the inequality follows from the fact that $\hat{\alpha}_i$ which is the sum of sizes of two Bloom filters is always at least α_i which is the sum of the sizes of the two sets being stored in those Bloom filters.

In order to prove (2), we need to compute the expected number of false positives in $\cap(\mathbf{U}, \mathbf{Out}(u_i))$ and $\cap(\mathbf{U}, \mathbf{In}(u_i))$, where u_i is the vertex of U processed at step i. A careful case analysis that separates the possible false positives into different categories based on which of the sets being intersected (if any) they belong to yields the result. We omit the proof due to lack of space. □

Now we show that the change in approximate conductance from one step to another has the same sign as the change in the exact conductance when certain conditions hold. The condition can be simply stated as follows: if the node added has small degree the conductance and the approximate conductance computed by *BloomConductance* either both increase or both decrease.

THEOREM 3.2. *Given a graph $G = (V, E)$ and a set of nodes $U \subset V$ as input to* BloomConductance(G, U)*, we assume that the algorithm considers the nodes of U in the order $u_1, u_2, \ldots$ and we denote by U_i the set of nodes $\{u_j : 1 \leq j \leq i\}$. Further, we denote by $\Delta(u_i)$ the error in the value of num in the ith iteration of the algorithm i.e.* $\Delta(u_i) = \mathsf{num}(\cap(\mathbf{U}_{i-1}, \mathbf{Out}(u_i))) + \mathsf{num}(\cap(\mathbf{U}_{i-1}, \mathbf{In}(u_i))) - \left|U_{i-1} \cap \overrightarrow{\Gamma}(u_i)\right| + \left|U_{i-1} \cap \overleftarrow{\Gamma}(u_i)\right|$.

- *If the conductance drops on considering a new node, u_i i.e. $\phi(U_i) < \phi(U_{i-1})$ and $|\overrightarrow{\Gamma}(u_i)| \left[\sum_{j=1}^{i-1} \Delta(u_j)\right] < \left[\sum_{j=1}^{i-1} |\overrightarrow{\Gamma}(u_j)|\right] \Delta(u_i)$ then the approximate conductance also drops i.e. $\hat{\phi}(U_i) < \hat{\phi}(U_{i-1})$, and*
- *if the conductance of a set rises on considering a new node, u_i i.e. $\phi(U_i) > \phi(U_{i-1})$ and $|\overrightarrow{\Gamma}(u_i)| \left[\sum_{j=1}^{i-1} \Delta(u_j)\right] > \left[\sum_{j=1}^{i-1} |\overrightarrow{\Gamma}(u_j)|\right] \Delta(u_i)$ then the approximate conductance calculation also rises i.e. $\hat{\phi}(U_i) > \hat{\phi}(U_{i-1})$*

We will need the following lemma which tells us that the decrease in computed conductance between one iteration of *BloomConductance* and the next can be upper bounded by the actual decrease in conductance if a certain technical condition holds.

LEMMA 3.3. *Given a graph $G = (V, E)$ and a set of nodes $U \subset V$ the approximate values $\hat{\phi}(U_i), 1 \leq i \leq |U|$ computed by* BloomConductance(G, U) *in its iterations have the property that*

$$\hat{\phi}(U_i) - \hat{\phi}(U_{i-1}) < \phi(U_i) - \phi(U_{i-1})$$

if and only if

$$|\overrightarrow{\Gamma}(u_i)|[\phi(U_{i-1}) - \hat{\phi}(U_{i-1})] < \Delta(u_i). \tag{3}$$

PROOF. The change in approximate conductance when u_i is added to the graph can be rewritten as

$$\hat{\phi}(U_i) - \hat{\phi}(U_{i-1}) = \frac{|\overrightarrow{\Gamma}(u_i)| - \hat{\alpha}_i - \hat{\phi}(U_{i-1})|\overrightarrow{\Gamma}(u_i)|}{\sum_{j=1}^{i}\left|\overrightarrow{\Gamma}(u_j)\right|}$$

If, and whenever, the condition (3) is true, this gives us:

$$\hat{\phi}(U_i) - \hat{\phi}(U_{i-1}) < \frac{|\overrightarrow{\Gamma}(u_i)| - \cap(i) - \phi(U_{i-1})|\overrightarrow{\Gamma}(u_i)|}{\sum_{j=1}^{i}\left|\overrightarrow{\Gamma}(u_j)\right|} = \phi(U_i) - \phi(U_{i-1}).$$

□

Now, since $\phi(U_i) - \hat{\phi}(U_i) = \frac{\sum_{j=1}^{i} \Delta(u_j)}{\sum_{j=1}^{i} \overrightarrow{\Gamma}(u_i)}$, Theorem 3.2 follows directly from Eq (1) of Proposition 3.1 and Lemma 3.3.

Discussion of Theorem 3.2

Consider the quantity $\Delta(u_i)$. This represents the "error" in estimating the number of outgoing edges after the ith node is added. When u_i enters the system some edges that were outgoing from $U_{i-1} = \{u_1, \ldots, u_{i-1}\}$ now become internal since they end in u_i (which is now part of the set U_i) and those that are outgoing from u_i into U_{i-1} cannot be added as outgoing edges from U_i, they are, in fact, internal to U_i. So these two sets of edges need to be estimated. The larger the size of these sets, the larger the error in this estimation, since Bloom filters give greater errors when we put more elements into the set. Theorem 3.2 makes this relationship more precise.

In Theorem 3.2 the main mathematical condition turns on the comparison between the ratios

$$\frac{|\overrightarrow{\Gamma}(u_i)|}{\left[\sum_{j=1}^{i-1} |\overrightarrow{\Gamma}(u_j)|\right]} \text{ and } \frac{\Delta(u_i)}{\left[\sum_{j=1}^{i-1} \Delta(u_j)\right]}.$$

Let us call the first one the *outdegree ratio* of the ith vertex and the second one the *intersection error ratio*.

To put it in words, Theorem 3.2 says that when the conductance drops on adding the ith vertex the ratio of the number outgoing edges of the ith vertex to the number of edges of the first $i - 1$ vertices must be smaller than the ratio of the error in estimating the number of edges that become internal on adding the ith vertex to the number of such edges for the first $i - 1$ vertices put together. This is a somewhat technical condition from which certain conclusions can be drawn. Let us investigate these.

If the ith vertex has a large outdegree relative to the vertices seen before, i.e., its outdegree ratio is high, then we expect the conductance to increase *and* if it shares few edges with the past vertices, i.e., its intersection error ratio is low, then Theorem 3.2 tells us that its approximate conductance will also rise. One way to ensure that a vertex with large outdegree has a low intersection error ratio is to *consider it earlier in the sequence, thereby ensuring that there are very few vertices in U_{i-1} for it to have intersections with.* So, if we have the freedom to choose the order in which the vertices are to be considered, it would make sense to consider vertices with high outdegree earlier. In other words the main pathological case occurs when a node with a large outgoing degree is considered late in the sequence, causing the exact conductance to grow. Being considered late in the sequence makes its intersection error ratio high and the condition of Theorem 3.2 gets violated and so there is no guarantee of the approximate conductance increasing. In Sec 4 we will see that such pathological cases occur rarely in real data sets.

3.3 The streaming scenario

In a streaming scenario as the community we are tracking evolves, some nodes enter and others leave. In such a situation it does not make sense to recompute conductance for the entire set of nodes from scratch since the change in the set of nodes considered could be quite small compared to the size of the set. In this section we leverage the large overlap from one time window to another to give a faster algorithm for computing the conductance of the set of nodes that occur in a data stream.

In the context of Twitter where our communities are defined as sets of users tweeting on a particular topic (or hashtag), we say formally that given a window of size β and a time increment of size α, at time t we consider those nodes that have talked about the topic between time $t - \beta$ and t. In the next step we will consider those nodes that have talked on the topic between time $t - \beta + \alpha$ and $t + \alpha$. In most cases, since α will be much smaller than β (say α is 1 hour and β is 24 hours), the difference between the two sets being considered in two successive steps is expected to be very small. In this scenario we would ideally like to add the small set of new nodes and delete the small set of nodes that are no longer to be considered when the window slides ahead.

The algorithm *BloomConductance* does not work in such a case since it relies only on the BloomGraph representation of the graph G and on Bloom filters to store the *nodes* of the subgraph induced by U. Consider the scenario where as time moves forward, a node $u \in U$ has to be deleted from the set of nodes U under consideration. In order to update the conductance we have to delete the set $\overrightarrow{\Gamma}(u) \setminus U$ which is the same as $\overrightarrow{\Gamma}(u) \cap \overrightarrow{\Gamma}(U)$. To facilitate this operation we would have to store the $\overrightarrow{\Gamma}(U)$ in a Bloom filter and delete those elements of $\overrightarrow{\Gamma}(u)$ that are also present in $\overrightarrow{\Gamma}(U)$. However $\overrightarrow{\Gamma}(U)$ may be a multiset, i.e., there may be a vertex w which has an incoming edge from u as well as from some other vertex $v \in U$. Even if deletion were possible, deleting w from the Bloom filter storing $\overrightarrow{\Gamma}(U)$ would remove w completely since storing multiple instances of the same value is not possible in a Bloom filter and, therefore, we would lose information about the edge (v, w).

In order to handle this issue, we will introduce what we call the *edge BloomGraph* of G, a structure in which the edges incident to each node are stored in Bloom filters of fixed size. To store an edge in a Bloom filter we need to give it an id. For this we use Cantor's Pairing function, a method for mapping a pair of integers to an integer [9]. The function is defined as follows

$$\pi(k_1, k_2) \triangleq \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2,$$

and has the property that it maps each pair to unique integer. For an edge $u \to v$ in our setting, we use the node ids of the users u and v, which are integers, as k_1 and k_2 and obtain $\pi(k_1, k_2)$ as the integer id of the edge. This id is then inserted into a counting Bloom filter that is supposed to store an edge set of which the edge $u \to v$ is an element.

DEFINITION 3.2. *Given a directed graph $G = (V, E)$ and a positive integer parameter m, the* edge BloomGraph $\mathfrak{B}(G, m, k)$ *of G is the set of Bloom filters $\{\widehat{\mathbf{In}}_G(u), \widehat{\mathbf{Out}}_G(u) : u \in V\}$, where $\widehat{\mathbf{In}}_G(u)$ is a Counting Bloom filter of size m that stores the set $\{\pi(v, u) : \forall v \in V$ and $(v, u) \in E\}$ and $\widehat{\mathbf{Out}}_G(u)$ is a Bloom filter of size m that stores the set $\{\pi(u, v) : \forall v \in V$ and $(u, v) \in E\}$, where $\pi(\cdot, \cdot)$ is the Cantor's pairing function. For each counting Bloom filter we use k bits at each location to store the number of hashes at that location. We refer to $\widehat{\mathbf{In}}_G(u)$ and $\widehat{\mathbf{Out}}_G(u)$ as* neighbourhood edge filters. *When the graph G is clear from the context we will drop the subscript and simply write $\widehat{\mathbf{In}}(u)$ and $\widehat{\mathbf{Out}}(u)$. Additionally we also store $|\overleftarrow{\Gamma}(u)|$ and $|\overrightarrow{\Gamma}(u)|$ for every $u \in V$.*

Our streaming algorithm will also need to store three edge sets associated with a set of nodes.

DEFINITION 3.3. *Given a directed graph $G = (V, E)$ and a positive integer parameter m, a set of nodes $U \subseteq V$ has three counting Bloom filters of size m associated with it. These are called $\widehat{\mathbf{Int}}(U)$, $\widehat{\mathbf{Ext}}_i(U)$ and $\widehat{\mathbf{Ext}}_o(U)$ and store the sets $\{(u, v) : u, v \in U\}$, $\{(u, v) : u \notin U, v \in U\}$ and $\{(u, v) : u \in U, v \notin U\}$ respectively, and are called the* BloomEdgeSets *associated with U. For brevity we will sometimes use the notation $\mathbb{BE}(U)$ to denote the BloomEdgeSets of U.*

With this representation in hand we present an algorithm to calculate conductance of an evolving set of nodes. In Figure 3 we present the description of an algorithm we call *BloomConductance$_s$* (*s* for streaming). We assume that this algorithm is given the BloomEdgeSets of a set of nodes U, along with two sets of nodes Λ_1 and Λ_2 where $\Lambda_1 \subset U$ and $\Lambda_2 \cup U = \emptyset$. In other words we assume that prior to the call of this function our set of nodes was U and some nodes from this are being deleted while some are being added. The net deletion is given by the set of nodes Λ_1 and the net addition is given by the set of nodes Λ_2. We assume that the node ids of the nodes in these sets are presented to the algorithm as input. The edge Bloomgraph $\mathfrak{B}(G, m, k)$ of the background graph G is assumed to be available throughout.

The algorithm works by using $\mathfrak{B}(G, m, k)$ to create the BloomEdgeSets of Λ_1 and Λ_2. This is done by a function called *GetBloomEdgeSets*. Then it takes the three sets of BloomEdgeSets (those belonging to U, Λ_1 and Λ_2) and uses them to create the BloomEdgeSets of $U'' = U \setminus \Lambda_1 \cup \Lambda_2$. This is done by first computing the BloomEdgeSets of $U' = U \setminus \Lambda_1$ using a function called *BloomSetMinus* and then computing the BloomEdgeSets of $U'' = U' \cup \Lambda_2$ using a function called *BloomUnion*. Once the BloomEdgeSets of this node set are available, calculating its conductance is easy.

```
Algorithm **BloomConductance**$_s(G, \mathbb{BE}(U), \Lambda_1, \Lambda_2)$
    $\mathbb{BE}(\Lambda_1) \leftarrow$ **GetBloomEdgeSets**$(\mathfrak{B}(G, m, k), \Lambda_1)$
    $\mathbb{BE}(\Lambda_2) \leftarrow$ **GetBloomEdgeSets**$(\mathfrak{B}(G, m, k), \Lambda_2)$
    /* Get the three BloomEdgeSets for each of the disjoint collection of nodes. */
    $\mathbb{BE}(U') \leftarrow$ **BloomSetMinus**$(\mathbb{BE}(U), \mathbb{BE}(\Lambda_1))$
    /* Compute the BloomEdgeSets of $U' = U \setminus \Lambda_1$. */
    $\mathbb{BE}(U'') \leftarrow$ **BloomUnion**$(\mathbb{BE}(U'), \mathbb{BE}(\Lambda_2))$
    /* Compute the BloomEdgeSets of $U'' = U \setminus \Lambda_1 \cup \Lambda_2$. */
    Set Conductance$(U'') \leftarrow \frac{\mathsf{num}(\widehat{\mathbf{Ext}_o}(U''))}{\mathsf{num}(\widehat{\mathbf{Int}}(U''))+\mathsf{num}(\widehat{\mathbf{Ext}_o}(U''))}$
    /* Compute conductance once we have $\mathbb{BE}(U'')$. */
```

Figure 3: *BloomConductance$_s$* run on subsets U, Λ_1 and Λ_2.

We omit a formal description of the procedures *GetBloomEdgeSets*, *BloomSetMinus* and *BloomUnion* due to space constraints. The first of these, *GetBloomEdgeSets*, incrementally builds the BloomEdgeSets of a set of nodes by considering the two edge Bloom filters associated with each node in the edge BloomGraph, taking care to determine which edge falls in which of the three BloomEdgeSets.

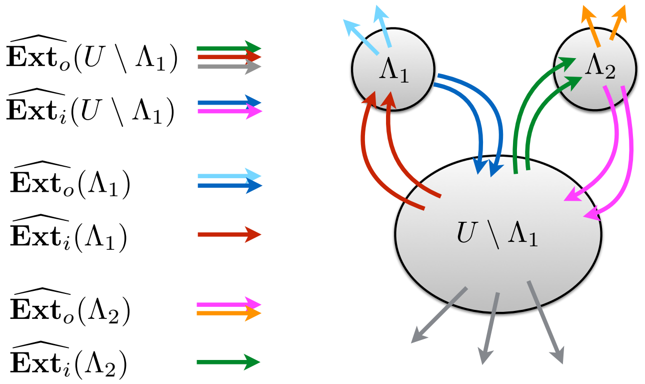

$\widehat{\mathbf{Ext}}_o(U \setminus \Lambda_1)$
$\widehat{\mathbf{Ext}}_i(U \setminus \Lambda_1)$
$\widehat{\mathbf{Ext}}_o(\Lambda_1)$
$\widehat{\mathbf{Ext}}_i(\Lambda_1)$
$\widehat{\mathbf{Ext}}_o(\Lambda_2)$
$\widehat{\mathbf{Ext}}_i(\Lambda_2)$
Λ_1
Λ_2
$U \setminus \Lambda_1$

Figure 4: The internal and external edge sets of U and $U \setminus \Lambda_1$ and $U \setminus \Lambda_1 \cup \Lambda_2$ can be easily computed from the internal and external edge sets of U, Λ_1 and Λ_2.

To illustrate the working of *BloomSetMinus*, we discuss the example shown in Figure 4. The set of internal edges of $U \setminus \Lambda_1$ can be constructed from the set of internal edges of U by removing the internal edges of Λ and the edges that go between $U \setminus \Lambda_1$ and Λ_1. These latter two sets, depicted in red and blue must be then added to the outgoing and incoming (respectively) edges of $U \setminus \Lambda_1$. To complete the picture the outgoing and incoming edges of Λ_1 which were *not* shared with vertices of $U \setminus \Lambda_1$ must also be removed from the outgoing and incoming (respectively) edges of U to get the outgoing and incoming (respectively) edges of $U \setminus \Lambda_1$. Similarly, to understand the working of *BloomUnion* we note that edges which were outgoing from one of the sets being unioned and incoming to another (the edges coloured green and pink in the Figure 4) must become internal in the union.

4. EXPERIMENTAL EVALUATION

We conducted experiments using real-world graphs to evaluate the scalability and the approximation quality of BloomGraphs. We compared the performance against an alternative of maintaining the graph and topic-wise subgraphs using standard adjacency list representation and computing conductance directly using it. In this section, we describe our experimental framework in detail, and present our key results.

4.1 Setup

We implemented BloomGraphs in C as a single-threaded program, and run all our experiments on a server-class machine with 4 × Xeon(R) E5-2660 v2 @ 2.20GHz cores, 64GB RAM, and 225GB harddisk used for persistently storing BloomGraphs. Our Bloom filter implementation uses hash function of the form:

$$g(x) = h_1(x) + ih_2(x) \mod p,$$

where $h_1(x)$ and $h_2(x)$ are two weak, independent, uniform hash functions on the universe of numbers with 64 bits, with range $\{0, 1, 2, \ldots, p-1\}$, where p is the size of the Bloom filter. This form of hash functions have been shown to be useful in implementing effective Bloom filters with no loss in their asymptotic false positive probability [25]. All experiments were conducted with 3 randomly chosen hash functions. With the exception of streaming setting where we use 2-bit counting Bloom filters, we always use simple Bloom filters. We consider BloomGraphs with varying Bloom filter lengths from $10,000$ to $40,000$. Similarly for counting Bloom filters we use lengths ranging from $30,000$ to $60,000$. It is expected that BloomGraphs with larger Bloom filter lengths will naturally be better in estimating the graph characteristics.

No. of Users	$7,695,882$
Avg. outdegree	450
No. of Users who tweet at least once in the data collection period	$3,008,496$
No. of Hashtags (after filtering)	$8,793,155$
No. of Tweets (over filtered hashtags)	$220,012,557$
No. of Hashtags with at least 100,000 tweets	119

Table 1: Characteristics of Twitter Dataset

In a preprocessing step, we read the underlying social network and write its BloomGraph representation to disk. In other words, each adjacency list of the graph is stored as a Bloom filter. It should be noted that we have made no effort to optimize the storage of Bloom filters – for instance, it is possible that for vertices with very few outgoing edges, a fixed size of Bloom filter bit-vectors may be more space inefficient than the standard representation of adjacency vector. Although one can consider space efficient implementation of resulting sparse Bloom filters, we leave it for future work.

4.2 Datasets

Our primary dataset was a Twitter activity crawl spanning a period of one month (from 27th March 2014 to 29th April 2014), from about 10 million users. Since we used Twitter REST API to collect these tweets, we worked around the limit of $3,000$ tweets for each user imposed by the APIs by collecting the data at a regular 2-week frequency. Each crawl resulted in a corpus of 3TB, and in the end we obtained 260 million tweets. We also used the same Twitter APIs to collect the background social network of these users, and found a strongly connected component consisting of 7.7 million users, which we focus our experiments on. Table 1 summarizes some of the key characteristics of this dataset. For more details on the dataset see [7].

In our experiments, we work on topic-focused subgraphs derived from a set of 119 hashtags which have a support of at least $100,000$ tweets in our collection. We also manually selected the following representative four topics (hashtags) which have distinctly different dynamics in our Twitter stream to illustrate the behaviour of BloomGraphs:

#HappyEarthDay: a hashtag that corresponds to the annual Earth day events celebrated on April 22 all across the world. As expected, this hashtag has a very well defined peak in activity in days surrounding its actual date, and almost no activity on other times.

#FollowFriday: corresponds to a weekly event hosted at Twitter, where you can recommend your followers to follow more people. Thus, we can expect weekly peaks of this topic, around every Friday.

#Haiyan: was used for tagging events related to typhoon Haiyan which struck Phillippines in 2013. Due to the extensive damage it caused, even in 2014, in particular during our crawling period, there were a few users – particularly those involved in relief and fund-raising activities – regularly tweeting about it.

#News: is a generic tag which is used for any item that can be considered as a news item. It shows a regular, high activity except during weekends.

4.3 Accuracy of Conductance Estimation

Our first experiment is aimed at demonstrating the ability of BloomGraphs to estimate the trend of conductance values of topic subgraphs. For this purpose, we consider the topic subgraph formed using tweets in the last 24 hours after each hour – i.e., we process

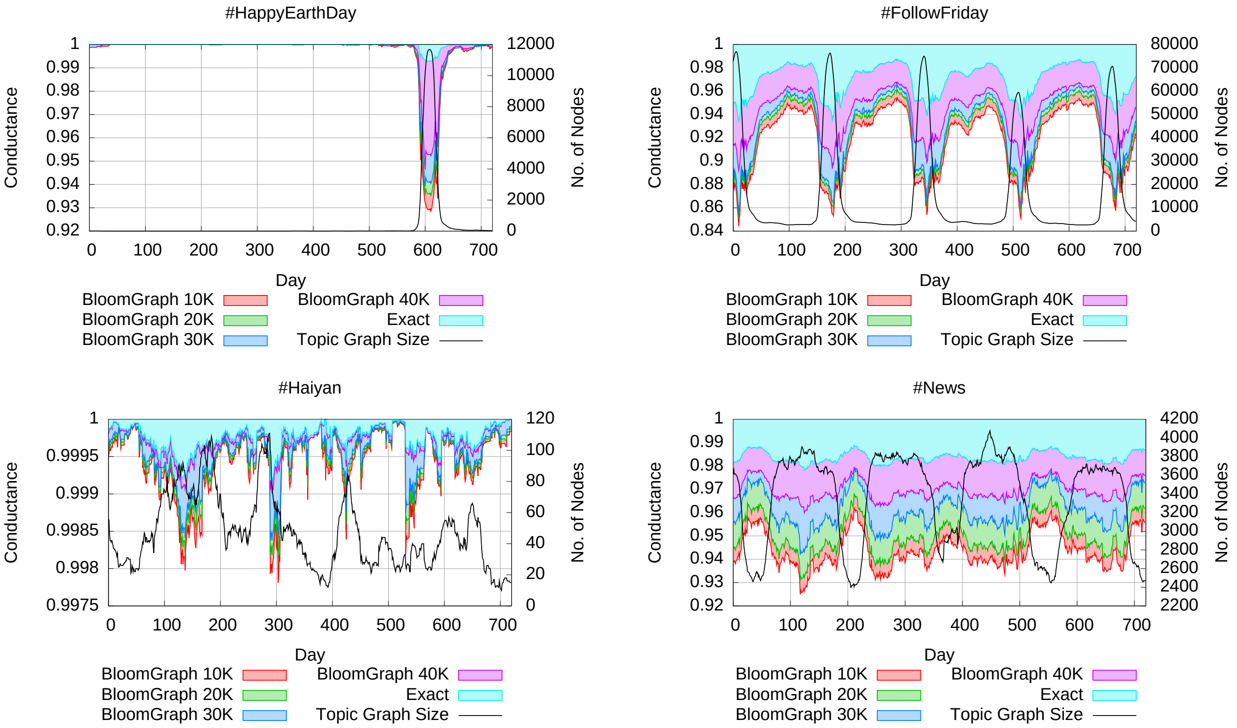


Figure 5: Accuracy of BloomGraphs in Tracking Conductance

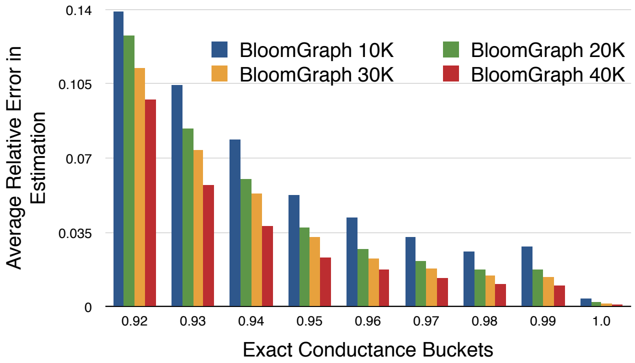


Figure 6: Average Relative Error in Conductance Estimation

tweets of last 24 hours, using BloomGraphs to continuously track the conductance of the topic subgraph.

The results for all four topics obtained using the 10K and 40K configurations of BloomGraphs, as well as using the *exact* computation method are shown in Figure 5. These plots also show the size of the topic subgraph (values correspond to the secondary y-axes of the plot). We can make the following observations:

- The conductance values estimated by BloomGraphs are consistently smaller than the exact conductance values (shown in Blue). This is simply an empirical confirmation of the Proposition 3.1,
- As expected, increasing the Bloom filter size improves the accuracy of estimation by BloomGraphs, which also follows from the Proposition 3.1, since the estimation accuracy we showed to be directly dependent on the false positive probability of the underlying Bloom filters,
- Irrespective of the estimation accuracy, the conductance scores reported by BloomGraphs show *the same trend* as the exact conductance scores. This is consistent with our Theorem 3.2 and subsequent Lemma 3.3.

These points clearly demonstrate that BloomGraphs are certainly valuable in settings where conductance scores of rapidly evolving topic subgraphs are used as a signal for their viral nature in the graph.

Furthermore, we can also see that the estimation accuracy of BloomGraphs is quite high for all the four topics considered. To investigate this further, we estimated conductance values with varying BloomGraph sizes over all 119 topics in our test-set. Figure 6 plots the average relative error in estimation as exact conductance values of the topic-graph vary. For ease of illustration, we have bucketized the conductance values at the granularity of 0.01, and plot from the smallest value ($= 0.92$) until the largest value ($= 1.0$). As seen from these results, estimations based on BloomGraphs are quite accurate even for small values of the conductance score. Although the relative error increases for smaller values, it should be noted that this is consistent with the claim that our estimation errors move in the same direction as the conductance scores.

Accuracy in Streaming Setting

Now, we turn our attention to the problem of conductance tracking in a *streaming scenario* where we would like to utilize the large amount of overlap in successive 24-hour time-windows(ref. 3.3). As we already described earlier, unlike the vertex-centric BloomGraphs that we used in the previous setting, we use the edge-centric algorithm $BloomConductance_s$ that uses edge BloomGraphs and BloomEdgeSets to computate conductance as tweets stream in.

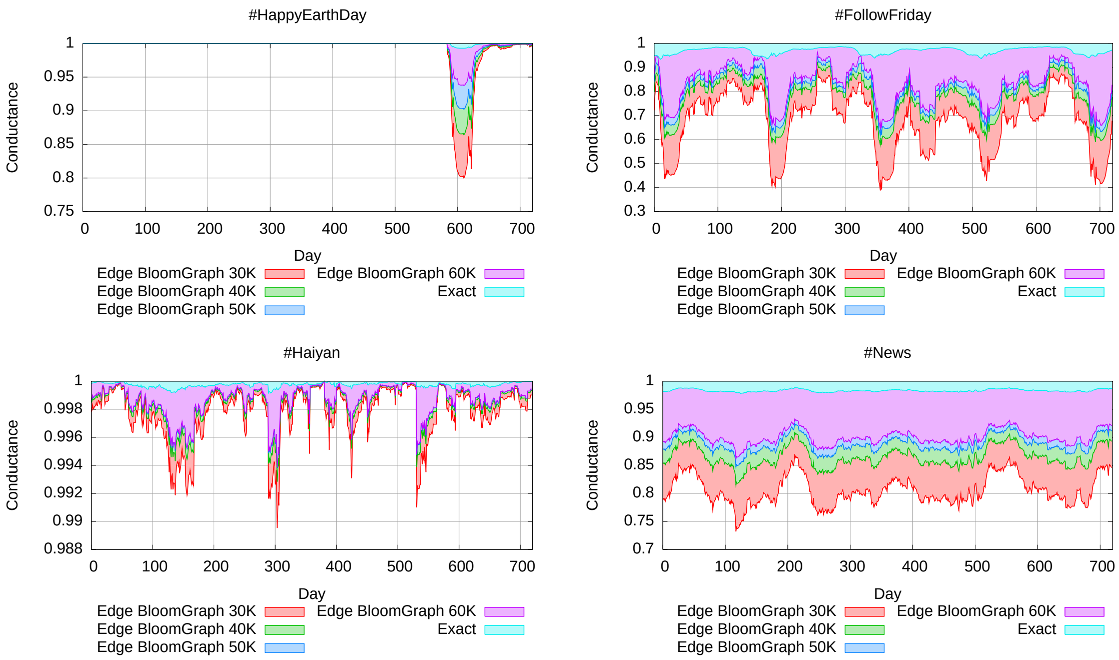


Figure 7: Accuracy of the Streaming Algorithm $BloomConductance_s$

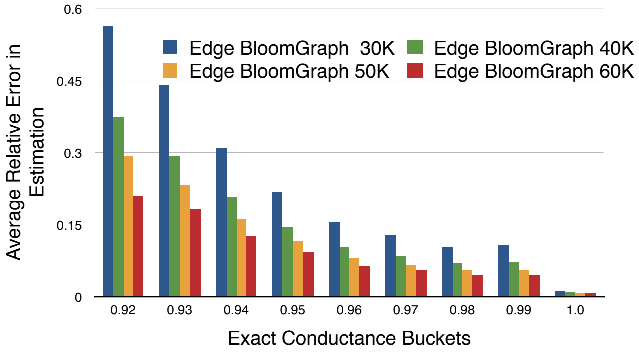


Figure 8: Average Relative Error of $BloomConductance_s$

Figure 7 plots the conductance estimates obtained when we use the $BloomConductance_s$ for tracking conductance of topic-subgraphs. When we compare these results with those in figure 5, it is immediately clear that the edge BloomGraph based method has higher error in estimating the conductance than the node-based BloomGraph-based algorithm. The plots in Figure 8 which show the average relative error of edge BloomGraphs in estimating conductance further strengthens this point. Despite this, one can also see that these errors are also always one-sided, and the relative error for the 40K configuration of Streaming BloomGraphs is typically under 30% for higher spectrum of exact conductance scores.

4.4 Efficiency

Experiment setup. We sorted the tweets by time and each tweet was fed to our system one by one. We randomly chose 10 topics for which to compute the conductance. In one run of the experiment, the conductance of exactly one topic was continuously updated for each tweet (provided the tweet corresponded to the topic) for a 24 hour window. The process was then repeated for the same topic by shifting the window by one hour. For each topic, there were around 500 such 24-hour windows, leading to a total of over 5000 runs.

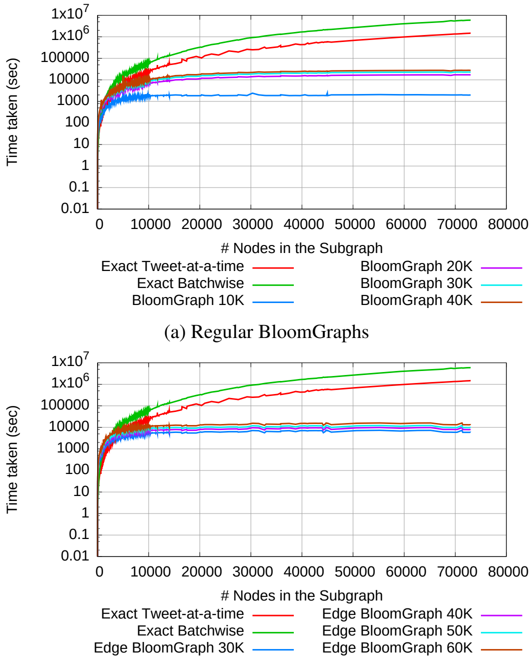
The performance of BloomGraphs were compared with the baseline method of computing conductance exactly after each tweet, using the adjacency list of the underlying social graph as well as the topic-focused subgraphs. Note that, for every tweet corresponding to the topic, either the originator of the tweet is already in the topic subgraph or needs to be added to it. For the former, the main computation is that of set membership, while for the latter, updating the conductance of topic-focused subgraph. We also did an exact *batch* computation by collecting all tweets in the 24 hour window and computing the conductance on the corresponding topic-focused subgraph.

Results. The results are shown in Figure 9 for *BloomConductance* and its streaming variants. The $X-$axis shows the size (number of nodes) of the topic subgraph and the $Y-$axis shows the total time taken to compute conductance in a 24-hour window.

Referring to Figure 9a, we find that computing conductance (one tweet at a time) using BloomGraphs is two orders of magnitude faster than the exact tweet-at-a-time computation, and almost 3-4 orders of magnitude faster than the exact batch computation, even if we use the 40K configurations. Next, we will consider the efficiency of the streaming variant $BloomConductance_s$ (see Figure 9b). The edge BloomGraph bases approach is slightly faster than the regular BloomGraph approach, and is not affected by the size of the subgraph under consideration. Further, it should be noted that in case of streaming algorithm $BloomConductance_s$, we can compute the conductance of topic graphs in a moving 24-hour window without having to store the tweets separately.

4.5 Memory Consumption

Since we use Bloom filters with 1- and 2-bits for each position, it is quite straight-forward to estimate the memory taken by each



(a) Regular BloomGraphs

(b) Edge BloomGraphs

Figure 9: Efficiency of Conductance Computation

BloomGraph. We illustrate this by considering 40K configuration: for 1-bit Bloom filter we would require about $(40000/8) \approx 5KB$ and for 2-bit counting Bloom filter about $(40000 * 2)/8 \approx 10KB$. While the space usage of regular BloomGraph remains approximately the same as a 1-bit Bloom filter, for edge BloomGraphs this is not the case. Note that for edge BloomGraphs we need to maintain the three BloomEdgeSets corresponding to each topic-graph (one for the internal and two for the external edgesets) leading to about $30KB$ of memory for each topic-graph.

This was easily confirmed by the following experiment: we simultaneously monitored $5,000$ topic-graphs in memory and obtained the memory footprint of the process from the USS metric returned by the *smem* utility. For BloomGraphs with $m = 40K$, the memory footprint was $26,388KB$ – consistent with our estimated value of $25,000KB$. Similarly, for *BloomConductance*$_s$, the memory footprint was reported as $169,166KB$ – again consistent with our estimated value of about $146,000KB$. In both cases, additional overheads are due to extra data-structures we maintain for analysing and computing the conductance value of each subgraph.

In summary, considering the need to compute conductance in a moving time-window over a stream of edge activations, edge BloomGraphs provide a competitive solution both in terms of their efficiency (see Figure 9b) as well as overall memory footprint.

4.6 Performance over Other Social Networks

Finally, we wanted to characterize the behavior of BloomGraphs in estimating the conductance of a subgraph in a large real-world social network, other than Twitter graph, as the size of the selected subgraph increases. For this purpose, we obtained the *soc-LiveJournal* dataset from Stanford SNAP website (`http://snap.stanford.edu`). This is a network consisting of close to 5 million users, with more 65 million edges connecting them. Due to lack of space, we do not provide detailed statistics of this dataset, and direct the interested reader to the SNAP website.

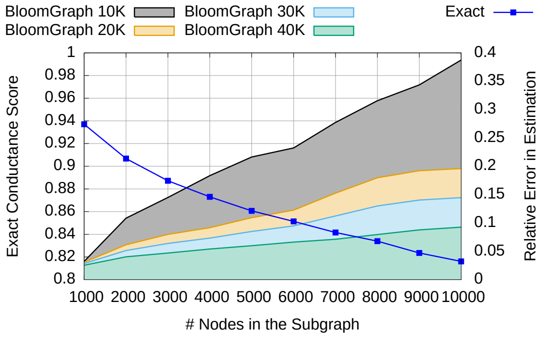


Figure 10: Accuracy of BloomGraphs on LiveJournal

This network, being a simple social network, differs quite significantly from the Twitter social graph and the topic-graphs considered so far. In LiveJournal graph there are no hashtags/topics on edges connecting two persons, and there are no dynamics in the network at all. Over this network, we randomly sampled a connected subgraph using a simple random-walk based sampling with uniform edge selection probabilities. We varied the size of the sampled subgraph between $1,000$–$10,000$ nodes in steps of $1,000$, and for each size we computed exact conductance score as well as the estimates using BloomGraph. The results, averaged over 5 random-walk runs, using different sizes of BloomGraphs are plotted in Figure 10. As these results show, BloomGraphs are quite effective even when the subgraph they operate on is quite large – especially if we use them in $40K$ configuration. In this case, the relative error in conductance scores is less than 10% even when the size of the subgraph is as large $10,000$. Although one sees larger error in the $10K$ configuration, as shown earlier, the conductance estimated by BloomGraphs follow the same trend as exact values.

5. RELATED WORK

Study of Social Networks

There exists significant literature on statistical trend prediction in streaming data, most notably for the Twitter data. In addition to the application-specific solutions there has been interest in tracking topics and events in these streams through an appropriate adaptation of statistical topic models (see, e.g., [1]). In this literature, the formulation of detecting bursts or spikes in topics [16] is relevant to our work. Goorha and Ungar study the spiking behavior of elements in an input set of elements [19]. Cataldi et. al. modeled the streaming elements as a graph and employed page-rank algorithm along with aging theory to predict spiking elements [10]. Mathioudakis and Koudas analyzed the sudden change in frequency patterns of these elements in conjunction with a reputation model for the origin of the streaming elements to make these predictions [31]. Becker et. al. discussed an online setting to identify events and the related tweets, but they did not make predictions for the future [5]. Our work in this paper can be seen as complementary to this line of research since we focus on the aspect of computational efficiency of computing the conductance, a popular metric widely used for studying the nature of communities in large networks.

Conductance

Conductance is a measure of how “expansive” or “close knit” a set of vertices of a graph is: higher conductance implies more outward connections for any cluster of vertices, lower conductance implies a more inward-looking cluster. As a result this quantity has very naturally found great applications in graph clustering [24]. A direct method of clustering involving computing the graph conductance, i.e. the minimum conductance over all subsets of the vertex set, designating the vertex set S that achieves the minimum as a cluster and then iterating with the rest of the graph is infeasible since computing the graph conductance is NP-hard and in fact thought to be inapproximable to within a factor of $\Omega(\sqrt{\log\log|V|})$ [11]. The class of clustering algorithms based on spectral methods provide provable performance bounds with respect to the graph conductance measure, but even in their case the second largest eigenvalue of the Laplacian of the graph can only approximate the graph conductance (see, e.g., [23, Chap. 4]).

However, since the conductance of a cluster can be easily measured in a setting where the graph can be efficiently stored, conductance has been widely used to measure the quality of a variety of clustering algorithms (see e.g. [28] or, [21] for surveys of clustering algorithms that use conductance as a quality measure.) In fact, the clustering coefficient, another popular and natural measure of the goodness of clustering, has also been shown to be tightly related to the graph conductance for certain classes of networks, which reinforces the fundamental nature of conductance as a measure of cluster quality [18].

The connection of graph conductance with the mixing times of random walks was conclusively established by Jerrum and Sinclair [22]. This well understood connection was extended to more general diffusion processes on networks, specifically rumors, by Chierichetti et. al. [12]. This connection was put on a solid empirical basis by Ardon et. al. [2]. In this work the authors studied the evolution of a number of topics on Twitter over a period of 80 days and found that the emergence of virality was closely related to a change in the conductance value i.e. when a topic was about to go viral the conductance of the (evolving) set of nodes of the Twitter network talking about the node underwent a sharp dip. No such behaviour was observed for topics that remained non-viral. This observation was correlated by Weng et. al. who used a feature called the first surface, which is closely related to conductance, to predict the growth to virality of hashtags in Twitter [36, 37]. These and other researchers have attempted to characterise and predict virality in social systems but have paid little or no attention to the computational feasibility of their methods. Our work attempts to bring an algorithmic and systems flavour to this area with the long term goal of building real-time systems that can feasibly use the analytical ideas developed to detect and predict viral topics in a timely fashion.

Algorithms for Large Graphs

Dynamic sets are key to a wide range of applications in computer networks, probabilistic verification, bio-informatics, and social networks. In many of these applications, these sets are too large to fit in memory for any analysis. For static sets (e.g., a set of edges of a large graph), distributed representations have been explored. For instance, Mondal and Deshpande [33] proposed efficient replication techniques for distributed graph databases. This work minimized network bandwidth consumption but did not address storage requirements for the graph. For streaming graphs, different efficient solutions have been proposed to solve specific tasks. Becchetti et al. [4] proposed efficient algorithms for local triangle counting in large, dynamic graphs. Sarma et al. [35] studied space-efficient estimation of page-rank for graph streams. Demetrescu et al. [14] show space-pass trade-offs for shortest path problems in graph streams. These approaches are specific to individual problems and do not lead to a common, efficient representation that generalizes well across different tasks.

Bloom Filters

Bloom filters provide an efficient representation for set membership queries. While their original formulation [6] did not support element deletion, subsequent variants e.g., counting Bloom filters [17] allowed for efficient deletion. Similarly, some extensions of Bloom filters such as stable Bloom filters [15] and timing Bloom filters [32] worked for sliding windows over data streams. Guo et. al. proposed dynamic Bloom filters that employ multiple filters with increasing capacity to enable representing large graphs [20]. Asadi and Lin proposed Bloom filter chains to rapidly retrieval for real-time search on Twitter stream [3]. Their work does not directly apply for studying the graph characteristics or virality prediction. Chikhi and Rizk used Bloom filters for in-memory representation of genome sequence with an additional consideration for removing critical false positives [13]. While the structure considered in this paper is a graph-based model (specifically the de-Bruijn graph), the particular structure of the graph makes it possible for the authors to maintain only a labelled vertex set and deduce the edge set from the vertex labels, making it significantly less general than the setting we study in the current paper. Broder and Mitzenmacher provide a detailed survey of diverse applications that employ Bloom filters [8].

6. CONCLUSIONS AND FUTURE WORK

Motivated by the problem of maintaining the conductance of a large number of network communities that are evolving at a high rate as new edges are activated and new nodes join the community, we have proposed a simple graph storage framework called *BloomGraphs* which use the well known Bloom filter structure to store adjacency lists. We have shown theoretically that the error incurred by BloomGraphs in computing conductance is one sided and so they are effective in detecting a sharp drop in conductance, a phenomenon that has been demonstrated in the literature to be a key indicator of an ascent to virality. We have demonstrated the efficacy of BloomGraphs on communities of users tweeting the same hashtag in Twitter network. This is a particular definition of community but we feel that BloomGraphs could be used over a broad-range of definitions of a community.

BloomGraphs open quite a few directions of research to pursue within dynamic graph analysis. While the state-of-art community detection algorithms operate on a static graph, there is no practical solution for *maintaining* these communities as more edges and nodes stream into the graph. BloomGraphs can be used to track the conductance of these communities, and when their conductance scores indicate that the communities are no longer stable, we can trigger their repair or rerun the community finding on the entire graph. In addition, we also plan to explore more compact implementation of Bloom filters, the use of BloomGraphs in other graph algorithms, and in community detection itself.

Finally, we feel that computing and updating complex metrics like conductance over evolving graph structures *in real-time* is already very important in the area of predictive analytics over networked structures, and this importance will only grow. This poses major computational challenges and the computing community's response has been to throw more resources at these challenges. It is our contention that to build these real-time systems it s is crucial that we build compact structures, like BloomGraphs, that return reasonable approximate answers very quickly.

7. REFERENCES

[1] L. AlSumait, D. Barbará, and C. Domeniconi. On-line LDA: adaptive topic models for mining text streams with

applications to topic detection and tracking. In *Proc. 8th IEEE Intl. Conf. on Data Mining (ICDM 2008)*, pages 3–12, 2008.

[2] S. Ardon, A. Bagchi, A. Mahanti, A. Ruhela, A. Seth, R. M. Tripathy, and S. Triukose. Spatio-temporal and events based analysis of topic popularity in Twitter. In *Proc. 22nd ACM Intl. Conf. on Information and Knowledge Management (CIKM 2013)*, pages 219–228, 2013.

[3] N. Asadi and J. Lin. Fast candidate generation for real-time tweet search with bloom filter chains. *ACM Trans. Inf. Syst.*, 31(3):13, 2013.

[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proc. 14th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD 2008)*, pages 16–24, 2008.

[5] H. Becker, F. Chen, D. Iter, M. Naaman, and L. Gravano. Automatic identification and presentation of twitter content for planned events. In *Proc. 5th Intl. Conf. on Weblogs and Social Media (ICWSM '11)*, 2011.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[7] S. Bora, H. Singh, A. Sen, A. Bagchi, and P. Singla. On the role of conductance, geography and topology in predicting hashtag virality. arXiv:1504.05351 [cs.SI], April 2015.

[8] A. Z. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Math.*, 1(4):485–509, 2003.

[9] G. Cantor. Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen. *J. für die reine und angewandte Mathematik*, 77:258–262, 1874.

[10] M. Cataldi, L. Di Caro, and C. Schifanella. Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proc. 10th Intl. Workshop on Multimedia Data Mining (MDMKDD '10)*, page Art. no. 4, 2010.

[11] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15:94–114, 2006.

[12] F. Chierichetti, S. Lattanzi, and A. Panconesi. Almost tight bounds for rumour spreading with conductance. In *Proc. 42nd ACM Symp. Theory of Computing*, pages 399–408, 2010.

[13] R. Chikhi and G. Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms Mol. Biol.*, 8:22, 2013.

[14] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *Proc. 17th Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA 2006)*, pages 714–723, 2006.

[15] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 25–36, 2006.

[16] Q. Diao, J. Jiang, F. Zhu, and E. Lim. Finding bursty topics from microblogs. In *Proc. 50th Ann. Meeting of the Assoc for Computational Lingustics (ACL 2012)*, volume 1, pages 536–544, 2012.

[17] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[18] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proc. 18th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD 2012)*, pages 597–605, 2012.

[19] S. Goorha and L. H. Ungar. Discovery of significant emerging trends. In *Proc. 16th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD 2010)*, pages 57–64, 2010.

[20] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo. The dynamic bloom filters. *IEEE Trans. Knowl. Data Eng.*, 22(1):120–133, 2010.

[21] S. Harenberg, G. Bello, L. Gjeltema, S. Ranshous, J. Harlalka, R. Seay, K. Padmanabhan, and N. Samatova. Community detection in large-scale networks: a survey and empirical evaluation. *WIREs Comput Stat*, 6:426–439, 2014.

[22] M. R. Jerrum and A. J. Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18:1149–1178, 1989.

[23] R. Kannan and S. Vempala. *Spectral Algorithms*. NOW, Delft, Netherlands, 2009.

[24] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, May 2004.

[25] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better bloom filter. In *Proc. 14th Annual European Symp. on Algorithms (ESA '06)*, pages 456–467, 2006.

[26] R. Krikorian. New tweets per second record, and how! Twitter Engineering Blog, 16th August 2013. https://blog.twitter.com/2013/new-tweets-per-second-record-and-how.

[27] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[28] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In *Proc. 19th Intl. Conference on World Wide Web (WWW '10)*, pages 631–640, 2010.

[29] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times*. AMS, 2009.

[30] L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. *Random Struct. Algor*, 4(4):359–412, 1993.

[31] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *Proc. ACM SIGMOD International Conference on Management of Data, (SIGMOD '10)*, pages 1155–1158, 2010.

[32] A. Metwally, D. Agrawal, and A. El Abbadi. Duplicate detection in click streams. In *Proc. 14th Intl. Conf. on World Wide Web (WWW 2005)*, pages 12–21, 2005.

[33] J. Mondal and A. Deshpande. Managing large dynamic graphs efficiently. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 145–156, 2012.

[34] J. Ronson. How One Stupid Tweet Blew Up Justine Sacco's Life. New York Times Magazine, Feb 2015. http://www.nytimes.com/2015/02/15/magazine/how-one-stupid-tweet-ruined-justine-saccos-life.html.

[35] A. D. Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. In *Proc. 27th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS 2008)*, pages 69–78, 2008.

[36] L. Weng, F. Menczer, and Y.-Y. Ahn. Predicting successful memes using network and community structure. In *8th Intl. AAAI Conference on Weblogs and Social Media (ICWSM 2013)*, 2013.

[37] L. Weng, F. Menczer, and Y.-Y. Ahn. Virality prediction and community structure in social networks. *Sci. Rep.*, 3:2522, 2013.