STRIP: Stream Learning of Influence Probabilities

Konstantin Kutzkov[1] Albert Bifet[2] Francesco Bonchi[2] Aristides Gionis[3]

[1]IT University of Copenhagen
Copenhagen, Denmark
konk@itu.dk

[2]Yahoo! Research
Barcelona, Spain
{abifet,bonchi}@yahoo-inc.com

[3]Aalto University and HIIT
Espoo, Finland
aristides.gionis@aalto.fi

ABSTRACT

Influence-driven diffusion of information is a fundamental process in social networks. Learning the latent variables of such process, i.e., the influence strength along each link, is a central question towards understanding the structure and function of complex networks, modeling information cascades, and developing applications such as viral marketing.

Motivated by modern microblogging platforms, such as `twitter`, in this paper we study the problem of learning influence probabilities in a data-stream scenario, in which the network topology is relatively stable and the challenge of a learning algorithm is to keep up with a continuous stream of tweets using a small amount of time and memory. Our contribution is a number of randomized approximation algorithms, categorized according to the available space (superlinear, linear, and sublinear in the number of nodes n) and according to different models (landmark and sliding window). Among several results, we show that we can learn influence probabilities with one pass over the data, using $\mathcal{O}(n \log n)$ space, in both the landmark model and the sliding-window model, and we further show that our algorithm is within a logarithmic factor of optimal.

For truly large graphs, when one needs to operate with sublinear space, we show that we can still learn influence probabilities in one pass, assuming that we restrict our attention to the most active users.

Our thorough experimental evaluation on large social graph demonstrates that the empirical performance of our algorithms agrees with that predicted by the theory.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - *Data Mining*

Keywords

Social network analysis, Social Influence, Streaming, Randomized approximation algorithms.

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

1. INTRODUCTION

Data from social networks and social media is generated continuously, creating streams that many applications need to process in real-time. Analysis of these social streams in real-time predicates the need for fast learning methods, which use a small amount of memory, and are capable of adapting to changes in the data distribution of the social network.

Diffusion of information drive by social influence is a fundamental process in social networks. Modeling and inferring latent information-influence variables is a central question towards understanding the structure and function of complex networks. Accordingly, a lot of research has been devoted in studying models of information diffusion and developing methods to learn their parameters. On the other hand, many of the proposed methods employ computationally-intensive techniques, such as EM-type schemes or approximation algorithms, which assume that one can operate with random access on the whole data, or can make many passes over the data. Obviously, such techniques are not suitable for nowadays applications that require processing and mining of large-scale data in continuous streams.

In this paper we propose STRIP, *a suite of streaming methods for computing the influence strength along each link of a social network*, that is, we learn the probability that each user influences each of his or her social contacts, friends, followers, etc. As a conceptual framework to compute the influence probabilities of the network edges, we are adopting the frequentist definition of Goyal et al. [13]. The brute-force computation of influence probabilities under this framework requires space that is proportional to the overall activity in the network, e.g., all the "tweets" that all users have posted.

We show how to efficiently estimate influence probabilities using much less space, and with one pass over the activity log. In particular, we express the space requirements of our algorithms as a function of the number of nodes in the network (n). This should be contrasted with other quantities in the network, such as the total number of edges (m), or the total number of actions performed at the network nodes. One should note that in modern social-microblogging applications, such as `twitter`, not only the number of users is expected to be much less than the size of the activity log, but also the set of users is much more stable while the actions performed by the users are continuous, rapid, and time-varying data streams. It is also worth noting that we aim at approximating a set of m probability values, one for each network edge, while expecting to keep a constant amount of memory for each network node. This is achieved by the

sketching nature of the STRIP framework: our algorithms maintain a constant-size sketch for each network node, and the influence probability of an edge (u, v) can be estimated directly by the sketches of the nodes u and v.

In addition to the algorithms we develop, we also present a number of theoretical results concerning the lower bounds for the required space complexity. In particular we study the *landmark model*, where the influence probabilities are computed with respect to the whole activity history of each user, and the *sliding-window model*, which are appropriate for forgetting data items that are not relevant, and thus adapting better to the behavioral changes occurring in the network.

Paper contributions and roadmap. The paper is organized in three main technical sections corresponding to the three memory settings: superlinear in the number of nodes n (Section 4), linear (Section 5), and sublinear (Section 6). Sections 4 and 5 are further divided in subsections presenting the STRIP methods based on the landmark model and the sliding-window model.

Our contributions are summarized as follows:

- Under the assumption that the whole social graph can be kept in memory, we present an algorithm for the landmark model (Section 4.1). For user-defined parameters $\varepsilon, \delta > 0$, the algorithm provides an (ε, δ)-approximation of the influence probability for *all* edges such that the quality of the estimates does not depend on the influence probability. Adjusting in a suitable way the exponential histogram technique, we extend the algorithm to estimate influence probabilities over time-based sliding windows (Section 4.2).

- For the setting where we are not able to keep the social graph in main memory we prove a lower bound showing that one cannot obtain an arbitrarily good approximation using sublinear memory for all edges, formalizing the intuition that it is difficult to estimate the influence probabilities among less active users in a streaming setting (Section 5.1). Then we present an algorithm whose space complexity is within a logarithmic factor of the lower bound building upon the Minwise sampling approach (Section 5.2). We extend the algorithm to the sliding window model (Section 5.3).

- For truly large graphs, when one needs to operate with sublinear space, we need to restrict our estimation of influence probabilities only to active users. More precisely, motivated by observations on real data, we assume that user activity adheres to Zipfian distribution with parameter z, i.e., assuming users are sorted in decreasing order according to their activity, for a stream of s actions in total the ith user performs $\frac{s}{\zeta(z) i^z}$ actions. We give an algorithm estimating the diffusion probabilities among the most active $b = o(n)$ users for the landmark model and sketch how it can be extended to the the sliding window model in Section 6.

We present some background and related work in Section 2. In Section 3 we give some preliminaries. We perform and discuss an empirical validation of the new methods proposed in Section 7, and finally Section 8 concludes the paper.

To the best of our knowledge this is the first work on learning influence probabilities from data streams.

2. BACKGROUND AND RELATED WORK

Stream learning. Data streams are large read-once sequences of data that change with time. Actions of users in social networks are an example of data streams. Data streams are usually so large that any exact computation is prohibitive in terms of memory and time. Thus stream learning methods are becoming popular since they allow real-time analytics on evolving data, under restrictions of time and memory. These methods usually rely on approximate algorithms that can obtain large gains in memory and time complexity, giving away little accuracy, by using sketch structures [6, 1], or applying sampling techniques, or a combination of both.

Learning influence probabilities. Detecting and estimating social influence strength among the users of social networks, is becoming a hot research topic in the computational social science as well as in the marketing literature. The amount of interest that this computational problem is attracting is justified by the great business potentialities of applications such as viral marketing, for which estimating influence strength is a needed preliminary step.

Given a social network, whose nodes are users and arcs represent social relations among the users, we can associate each arc (u, v) with a probability p_{uv} representing the strength of influence exerted by u on v. Or in other terms, the probability that a tweet posted by u will be "retweeted" by v. In this setting, a basic computational problem is that of selecting the set of users to be targeted by the viral marketing campaign: those are the users more likely to generate a large viral cascade. The first algorithmic treatment of the problem was provided by Domingos and Richardson [10]. Later, Kempe et al. [16] introduced *influence maximization* as a discrete optimization problem: given a budget k, find the set of k nodes that maximizes the expected number of active nodes at the end of the process. The activation of nodes is governed by a probabilistic propagation model. For instance, in the *Independent Cascade* propagation model, when a node u first becomes active, say at time t, it has one chance of influencing each inactive neighbor v with probability p_{uv}. If the tentative succeeds, v becomes active at time $t + 1$. Following this seminal work [16], considerable effort has been devoted to develop methods for improving the efficiency of influence maximization [18, 7, 14]. The majority of this literature assumes the input social graph has already the influence probabilities associated to links, and does not address how to compute them.

Saito et al. [21] were the first to study the problem of learning the probabilities for the independent cascade model from a set of past observations, formalizing it as likelihood maximization and applying Expectation Maximization (EM) to solve it. However, the iterative nature of EM methods is really not suited for stream processing. In this paper we adopt instead the simpler frequentist definition of Goyal et al. [13]. The learning procedure defined in [13] assumes that the input propagation data is stable and sorted by the item and then by time. Of course the real stream of events does not come sorted by any criteria except time, so that the propagations corresponding to different items arrive intertwined. The learning algorithm needs two scans of this fixed and sorted database. Moreover, it keeps in memory the whole propagation of an item, plus one counter for each node and a couple of counters for each link.

3. PRELIMINARIES

We consider a social network, represented as a directed graph $G = (V, E)$ with users corresponding to vertices and edges to social connections, such that the edge (u, v) denotes that v *follows* u. The maximum in-degree of a vertex in G is denoted by Δ. Let A be a *set of actions*. The input is provided as *stream* $\mathcal{S}$ of triples (u, a, t_u). Each triple denotes that user u performed action $a \in A$ at time t_u. The number of actions in $\mathcal{S}$ is denoted as s. An action *a propagates within* τ *time units* from user u to user v if $(u, v) \in E$, and a is performed by u and then by v within τ time units, i.e., there exist triples $(u, a, t_u), (v, a, t_v) \in \mathcal{S}$ with $0 < t_v - t_u \leq \tau$. We are interested how influential is a given user u, i.e., how probable is that actions propagate from user u to a user v.

We consider two standard streaming models: In the *landmark model* one is interested in the whole history of the stream starting from a given time point to the present. Let A_u denote the set of actions performed by user u. We also define A^{τ}_{u2v} to be the set of actions propagated from u to v within τ time units, and $A_{u|v}$ the set of actions performed by either u or v. We set $\max(A, \tau)$ to be the maximum number of actions that can be performed by a user within τ time units. To estimate the influence probability p_{uv} between the users u and v, we use the so-called Jaccard model, proposed by Goyal et al. [13], and define $p_{uv} = \frac{s(A^{\tau}_{u2v})}{s(A_{u|v})}$, where, hereinafter, $s(A)$ denotes the size of a set A. In *time-based sliding windows* only the most recent actions are considered for a user-defined time threshold. Denote the window by W and the set of actions performed within the window by A^W. The notation for the actions performed by a user within the window W extends in the obvious way the notation for the landmark model. We set $w = \arg\max_{u \in V} s(A^W_u)$.

Note that if we are able to keep in main memory the whole stream, we can easily compute the influence probabilities defined above. However, for high-frequency streams of very large volume this is not possible and we thus need to develop stream-processing algorithms computing approximate estimates of the influence probabilities. Naturally, our solution extends techniques developed for streaming algorithms. We review the relevant background below.

Probabilistic approximation. We say that an algorithm returns an (ε, δ)-*approximation* of some quantity q, if it returns a value $\tilde{q}$, such that $(1 - \varepsilon)q \leq \tilde{q} \leq (1 + \varepsilon)q$, with probability at least $1 - \delta$ for any $0 < \varepsilon, \delta < 1$.

Min-wise independent functions. A family $\mathcal{F}$ of functions from U to Z is k-*wise independent* if for $f : U \to Z$ chosen uniformly at random from $\mathcal{F}$

$$\Pr[f(u_1) = c_1 \wedge f(u_2) = c_2 \wedge \cdots \wedge f(u_k) = c_k] = z^{-k},$$

for $z = s(Z)$, distinct $u_i \in U$ and any $c_i \in Z$ and $k \in \mathbb{N}$.

A family $\mathcal{H}$ of functions from U to a finite totally ordered set S is called (α, k)-*min-wise independent* if for any $X \subseteq U$ and $Y \subseteq X$, with $|Y| = k$ and $0 < \alpha < 1$, for a function h chosen uniformly at random from $\mathcal{H}$ it holds

$$\Pr[\max_{y \in Y}\{h(y)\} < \min_{z \in X \setminus Y}\{h(z)\}] = (1 \pm \alpha)\frac{1}{\binom{|X|}{k}}.$$

We use the notation $h : U \to [0, 1]$ to denote that h maps U to a finite subset D of $[0, 1]$. For h being pairwise independent and $|D| = |U|^3$, the probability of collision, i.e., $h(u_1) = h(u_2)$ for $u_1, u_2 \in U$, $u_1 \neq u_2$, is at most $1/|U|$. We thus assume that $h : U \to [0, 1]$ is injective with high probability and $h(u)$ can be described using $\mathcal{O}(\log |U|)$ bits. For the analysis of our algorithms we will use the following

FACT 1. *Let c_1, c_2, c_3, c_4 be constants larger than 1. For any $0 < \varepsilon < 1$ there exists an $\varepsilon' = \Theta(\varepsilon)$ such that*

$$\frac{(1 + c_1\varepsilon')^{c_2}}{(1 - c_3\varepsilon')^{c_4}} \leq 1 + \varepsilon, \text{ and } \frac{(1 - c_1\varepsilon')^{c_2}}{(1 + c_3\varepsilon')^{c_4}} \geq 1 - \varepsilon.$$

Min-wise independent hashing. Min-wise independent permutations [4, 8] is a powerful technique for estimating the Jaccard similarity $J(A, B) = \frac{s(A \cap B)}{s(A \cup B)}$ between two sets A and B, subsets of a ground set U. In particular, let π be a random permutation of the elements in U. Define $\pi(A) = \min_{x \in A}\{\pi(x)\}$, the smallest element of A under the permutation π, and $\pi(B)$ similarly. It is easy to see that $\Pr[\pi(A) = \pi(B)] = J(A, B)$, namely, the indicator variable that the smallest elements in $\pi(A)$ and $\pi(B)$ are identical yields an unbiased estimator of the Jaccard similarity.

The approach can be generalized to estimating the fraction $\frac{s(\Omega(A \cup B))}{s(A \cup B)}$ for any efficiently computable predicate Ω on the elements in $A \cup B$. For instance, for a random permutation π, we can obtain an unbiased estimator of the influence probability p_{uv} by keeping track of $a_u = \arg\min_{x \in A_u}\{\pi(x)\}$ and $a_v = \arg\min_{x \in A_v}\{\pi(x)\}$, and then checking whether $a_u = a_v$ and a_v has been performed within τ time units of a_u. By the estimator theorem [19], with $\mathcal{O}(\frac{1}{\varepsilon^2 p_{uv}} \log \frac{1}{\delta})$ independent estimators one obtains an (ε, δ)-approximation of p_{uv}. The applicability of the approach on a streaming setting, follows from the fact that, as shown by Broder et al. [4], the random permutations can be replaced by efficiently computable hash functions $h : A \to [0, 1]$.

A modification of the approach stores the k smallest hash values instead of a single hash value and returns as an estimate $\frac{s(\Omega(\mathrm{Min}_k(A \cup B)))}{k}$, where $\mathrm{Min}_k(A \cup B)$ are the actions in $A \cup B$ with the k smallest hash values. By choosing Ω to be the function that returns the number of propagated actions in the sample, one can show that for $k = \mathcal{O}(\frac{1}{p\varepsilon^2})$ and an (α, k)-independent hash function, for some constant $\alpha < 1/2$, the scheme yields an $(1 \pm \varepsilon)$-approximation for edges with influence probability at least p with probability more than $1/2$. Standard application of Chernoff bounds yields that the median of $\mathcal{O}(\log \frac{1}{\delta})$ independent estimators is an (ε, δ)-approximation.

The latter scheme is more suitable for our purposes. Additionally, a recent improvement of the approach exponentially reduced the evaluation time of an (α, k)-independent hash function from $\mathcal{O}(k)$ to $\mathcal{O}(\log^2 k)$ [12]. For the theoretical analysis of our algorithms we use the following

FACT 2. *Let $\mathcal{A}^k$ be the set of size-k subsets of A and let more than $2/3$ of these satisfy a certain property ρ. Let $X \in \mathcal{A}^k$ be a set satisfying ρ. Then there exists a constant $\alpha < 1/2$ such that an (α, k)-independent hash function h maps the elements of X to the k smallest hash values with probability more than $1/2$. The function h can be described in $\mathcal{O}(k)$ machine words and evaluated in time $\mathcal{O}(\log^2 k)$.*

Sliding-time windows. Datar et al. [9] presented an algorithm for estimating the number of 1's in a stream of bits over time-based sliding windows. The algorithm relies on a data structure based on *exponential histograms*. We adjust the approach such that we can apply min-wise independent hashing over sliding windows. As a result we can keep

track of the minimum element, according to a random permutation π, in the sliding window. In the analysis of our algorithms we will use the following

FACT 3. *Let W be a time-based window and w be the number of actions performed within W. Keeping an exponential histogram with $\mathcal{O}(\log w/(\varepsilon))$ buckets for each user, one can compute the minimum hash value of at least $(1-\varepsilon)w$ of the actions in W.*

4. SUPERLINEAR SPACE

In this section we assume that the whole graph fits in memory and for each user we can store all actions performed within τ time units. We present algorithms for the landmark and the sliding-window model. The algorithms provide an (ε, δ)-approximation for the influence probability p_{uv} of all $(u, v) \in E$, and the complexity does not depend on p_{uv}.

4.1 Landmark model

The superlinear-space landmark-model algorithm, shown as Algorithm 1, works as follows. For each user u we keep a data structure Q_u recording the performed actions together with a hash table H_u recording the time each action in Q_u has been performed. For each directed edge (u, v) we keep a counter $c(A_{u2v})$ recording the number of actions propagated from u to v within time τ, respectively. For each user u we keep a sketch $\texttt{MinHash}_u$ of the actions. The sketches of the users u and v will be used to estimate $s(A_{u|v})$ [1]. For a pairwise independent hash function $h : A \to [0, 1]$ the sketch for user u consists of the k smallest hash values of the actions performed by u. For each incoming triple (u, a, t_u) we update Q_u and H_u by first deleting all actions performed more than τ time units ago and adding the pair (a, t_u). Then we evaluate the hash value of the action performed and update $\texttt{MinHash}_u$, i.e., we add $h(a)$ to u's sketch if it is smaller than the largest value or there are less than k hash values in the sketch. Next we check which of u's neighbors have already performed a within time τ and increment $c(A_{u2v})$. At the end we estimate $s(A_{u|v})$ as proposed in [1] by returning k/h_k, where h_k is the k-th smallest hash value in $\texttt{MinHash}_u \cup \texttt{MinHash}_v$.

THEOREM 1. *Let $G = (V, E)$ be a connected graph over n vertices and m edges, and $\mathcal{S}$ be a stream of s actions. There exists an algorithm computing an (ε, δ)-approximation of the influence probability of all edges in expected amortized time $\mathcal{O}(s(\Delta + \log \frac{n}{\delta}) + \frac{m}{\varepsilon^2})$ and space $\mathcal{O}(m + n(\max\{A, \tau\} + \frac{1}{\varepsilon^2} \log \frac{n}{\delta}))$ in only one pass over $\mathcal{S}$.*

PROOF. For a user u we keep a queue Q_u recording the performed actions together with the corresponding time as well as a hash table H_u containing the actions. For each user we also maintain the k actions with the smallest hash values seen so far. Clearly, Q_u and H_u consists of $\mathcal{O}(\max\{A, \tau\})$ pairs since we maintain the invariant that the first and last action in Q are preformed within τ time units. Also each pair is added and deleted exactly once to a given Q_u and H_u, thus each incoming action is processed in expected constant amortized time. We observe that storing the incident edges for each user u in a hash table, we can access the neighbors of a user u in expected time bounded by $\mathcal{O}(\Delta)$.

UPDATE maintains the k smallest hash values of actions performed by each user. One can use a priority queue, this

Algorithm 1: STRIP - SUPERLINEAR MEMORY

```
PROCEEDSTREAM
Input: Social graph G = (V, E), stream S of s actions,
       threshold τ, int k, pairwise independent
       h : A → (0, 1]
Load G in memory.
for (u, a, t) ∈ S do
    Q_u.enqueue(a, t).
    H_u.add(a, t)
    UPDATE(MinHash_u, h(a), k)
    while Q_u is not empty do
        (a_u, t_u) ← Q_u.last()
        if |t_u − t| > τ then
            Q_u.remove()
            H_u.remove(a, t))
        else
            break;
    for (v, u) ∈ E do
        if H_v.contains(a, t_v) and t_u − t_v ≤ τ then
            c(A_v2u)++

ESTIMATEUNIONSIZE
Input: vertex u, vertex v, int k
MinHash ← MinHash_u ∪ MinHash_v
h_k ← the k-th smallest value in MinHash
return k/h_k

ESTIMATEINFLUENCEPROBABILITIES
Input: Social graph G = (V, E)
for (v, u) ∈ E do
    s̃(A_v|u) ←ESTIMATEUNIONSIZE(u, v, k)
    return c(A_v2u)/s̃(A_v|u)
```

however incurs an additional $\log k$-factor for the running time. Instead, for each user u we maintain the k-th smallest hash value $h^k_{\min}$. Then for an incoming triple (u, a, t) we check whether $h(a) < h^k_{\min}$ and if so, we store $h(a)$ in a auxiliary data structure $\texttt{MinHash}^{\text{aux}}_u$. Once there are k values in $\texttt{MinHash}^{\text{aux}}_u$, we find the median in $\texttt{MinHash}_u \cup \texttt{MinHash}^{\text{aux}}_u$ and update $\texttt{MinHash}_u$ to contain the k minimum hash values. The median can be found in $\mathcal{O}(k)$ time by a deterministic algorithm [2], thus we update $\texttt{MinHash}_u$ in constant amortized time. Looking-up in a hash table takes expected constant time. For each edge (u, v) ESTIMATE(u, v) computes h^k_{uv}, the k-th smallest hash value in $\texttt{MinHash}(u) \cup \texttt{MinHash}(v)$. As shown by Bar-Yossef et al. [1], for $k = \mathcal{O}(\frac{1}{\varepsilon^2})$ the value $1/h^k_{uv}$ is an $(1 \pm \varepsilon)$-approximation of $s(A_{u|v})$ with probability at least 2/3. Thus the median of $\log \frac{m}{\delta} = \mathcal{O}(\log \frac{n}{\delta})$ such estimators is an $(\varepsilon, \frac{\delta}{n})$-approximation of p_{uv} and by the union bound we have an (ε, δ)-approximation of the number of performed actions for all edges. The (ε, δ)-approximation of the influence probabilities then follows from Fact 1. □

4.2 Sliding window model

A straightforward extension of Algorithm 1 for the sliding window model is to combine it with the exponential histogram approach using $t = \mathcal{O}(\frac{1}{\varepsilon})$ buckets. In each bucket we will keep the k smallest hash values and at the end, using Fact 3, we will be guaranteed to have the k smallest hash values for at least $(1 - 2\varepsilon)$ of the actions performed by each user. However, we can make use of the fact that hash

values are random and thus it is very improbable that the k smallest actions will come all from the same bucket. We formalize this intuition in the next theorem and show that for a sufficiently random hash function it suffices to store only the smallest $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ hash values per bucket.

THEOREM 2. *Let $G=(V,E)$ be a connected graph over n vertices and m edges, $\mathcal{S}$ be a stream of s actions and W a time-based sliding window. There exists an algorithm computing an (ε,δ)-approximation of the influence probability of all edges over a time-based sliding window in expected amortized time $\mathcal{O}(s(\Delta+\log\frac{n}{\delta}\log\frac{1}{\varepsilon})+\frac{m}{\varepsilon^2})$ and space $\mathcal{O}(m\log w\frac{1}{\varepsilon}+n(\max\{A,\tau\}+\frac{1}{\varepsilon^2}\log^2(\frac{1}{\varepsilon})\log w\log\frac{n}{\delta}))$ in only one pass over $\mathcal{S}$.*

PROOF. We use a bit-counter over sliding windows to estimate $s(A^W_{u2v})$. In order to estimate $s(A^W_{u|v})$ we extend the exponential histogram approach as follows. Consider a given user u and assume that s/he performs w actions within the window. We maintain $t=\frac{1}{\varepsilon}$ buckets of width 2^i for $0\le i\le\log\frac{w}{2t}$, each keeping track of $2^i\le\varepsilon w/2$ actions.

Let $k=\mathcal{O}(\frac{1}{\varepsilon^2})$. Since h maps the w actions uniformly at random to values in $[0,1]$, we expect k hash values to be at most k/w. By Markov's inequality the probability that the k-th smallest value is larger than $3k/w$ is at most $1/3$. Next we bound the number of small hash values in a given bucket. For a bucket recording at most εw actions we expect $3k\varepsilon=\mathcal{O}(\frac{1}{\varepsilon})$ hash values in the bucket to be smaller than $3k/w$. For a fully random hash function, by Chernoff bounds the probability that the number of hash values smaller than $3k/w$ is more that $\mathcal{O}(k\varepsilon\log t)$ is $1/(8t)$. Thus, for uniformly distributed hash values in $[0,1]$, a fraction of $1/(8t)$ of the size-x subsets of the actions in a given bucket, for $x=\mathcal{O}(k\varepsilon\log t)=\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$, will constitute of actions with hash values all smaller than $3k/w$. Thus, by Fact 2, there exists constant $\alpha<1/2$ such that an (α,x)-independent hash function guarantees with probability $1/(6t)$ that the hash values for at most x actions in a given bucket are smaller than $3k/w$. By the union bound we find the k smallest hash values for the buckets of given width with probability at least $5/6$, thus the total space to compute the smallest $\mathcal{O}(\frac{1}{\varepsilon^2})$ hash values is $\mathcal{O}(\frac{1}{\varepsilon^2}\log^2(\frac{1}{\varepsilon})\log w)$. Each action is processed in amortized time $\mathcal{O}(\log\frac{1}{\varepsilon})$.

From Fact 3 it follows that we might not have a record for εw of the actions in A^W_u and A^W_v. In the worst case we thus have not considered $2\varepsilon s(A^W_{u|v})$ different actions and we compute an $(1\pm\varepsilon)$-approximation of the quantity $(1-2\varepsilon)s(A^W_{u|v})$ with probability at least $2/3$. Since the bit-counter for $s(A^W_{u2v})$ computes an $(1\pm\varepsilon)$-approximation of the number of actions propagated from u to v, by Fact 1 it follows that by rescaling ε, i.e., $\varepsilon:=\varepsilon/c$, for some constant $c>0$, we have an $(1\pm\varepsilon)$-approximation of the influence probabilities for an edge (u,v) with probability at least $2/3$. Thus, the success probability for detecting the k smallest hash values and obtaining an $(1\pm\varepsilon)$-approximation from them is $(5/6)\cdot(2/3)>1/2$. Taking the median of $\mathcal{O}(\log\frac{n}{\delta})$ estimates yields the claimed result. □

5. LINEAR SPACE

We now move to the more interesting case, presenting algorithms that require space linear in the number of vertices. Our algorithms are appropriate for dense networks, and when we assume that there is $\mathcal{O}(n)$ available space but not $\mathcal{O}(m)$, the so-called *semi-streaming* model [11]. From the empirical point-of-view it is known that social networks become denser over time [17] and it is conjectured that $m=\Omega(n^{1+\varepsilon})$ for some constant $\varepsilon>0$. Also, it is feasible to keep the number of actions for each user performed within τ time units only for relatively small τ. If one is interested in learning the probabilities over longer time intervals, one has to store a considerable fraction of the stream in memory.

5.1 Lower bound

Before providing our algorithms, we discuss that linear space in the number of nodes is necessary if one wants to learn the influence probabilities on all edges with certain accuracy. Intuitively, we need to store some information about each vertex in the social graph because the influence probability on a given edge (u,v) can be high even if the number of actions performed by u and v is very small.

The proof of the next theorem can be found in the Appendix. It uses a simple reduction from the BIT-VECTOR DISJOINTNESS problem for binary vectors over n bits which is known to have communication complexity of $\Omega(n)$ bits [15]. Note that the result holds for randomized algorithms even if they are allowed to make a constant number of passes on the data stream.

THEOREM 3. *Let $G=(V,E)$ be a connected graph over n vertices and m edges, and $\mathcal{S}$ be the action stream. Consider any randomized streaming algorithm $\mathcal{A}$ that makes a constant number of passes over E and $\mathcal{S}$. Assume that $\mathcal{A}$ distinguishes with probability more than 1/2 the following two cases (1) all edges have influence probability at most $1/(d-1)$, for any $d\ge 3$, vs. (2) there is an edge with influence probability $1/2$. Then $\mathcal{A}$ needs $\Omega(n)$ bits in expectation.*

5.2 Landmark model

Algorithm 2 is based on min-wise independent sampling. We assume $h:A\to[0,1]$ is t-wise independent, for t that will be specified later. For each user u we keep a sample $\texttt{MinHash}_u$ of the k actions with the smallest hash values performed by him/her together with the time-stamp the action was performed. For each incoming action (u,a,t) we evaluate $h(a)$ and update $\texttt{MinHash}_u$ to contain the k actions with smallest hash values. After processing the stream, for a given edge (u,v) we determine the (at most) k actions with the smallest hash values in $\texttt{MinHash}_u\cup\texttt{MinHash}_v$ and in $\mathrm{Prop}(\texttt{MinHash}^k_{u|v},\tau)$ we count how many of them propagated from u to v within τ time units.

THEOREM 4. *Let $G=(V,E)$ be a connected graph over n vertices and m edges, and $\mathcal{S}$ be a stream of s actions. There exists an algorithm returning an (ε,δ)-approximation of the influence probability p_{uv} of all arcs (u,v) with $p_{uv}\ge p$ in amortized time $\mathcal{O}((s\log^2(\frac{1}{\varepsilon p})+\frac{m}{\varepsilon^2 p})\log\frac{n}{\delta})$ and space $\mathcal{O}(\frac{n}{\varepsilon^2 p}\log\frac{n}{\delta})$ in one pass over $\mathcal{S}$.*

PROOF. Assume that h is t-wise independent, for t to be specified later. Consider an edge $(u,v)\in E$. We are interested in those sets $\texttt{MinHash}^k_{u|v}$ of k elements containing $(1\pm\varepsilon)p_{uv}k$ actions that propagated from u to v. The number of such sets follows hypergeometric distribution. We expect $p_{uv}k$ actions in $\texttt{MinHash}^k_{u|v}$ to have propagated from u to v. With some algebra we bound the variance of the random variable counting the number of propagated actions in $\texttt{MinHash}^k_{u|v}$. By Chebyshev's inequality it follows that for

$k = \mathcal{O}(\frac{1}{\varepsilon^2 p})$ at most 1/3-rd of the k-size subsets will not yield a $(1 \pm \varepsilon)$-approximation of p_{uv}. From Fact 2 there exists a constant $\alpha < 1/2$ such that for h being (α, k)-independent, with probability at least 2/3 the k smallest hash values correspond to a set of size k which provides an $(1 \pm \varepsilon)$-approximation. A standard application of Chernoff's inequality and the union bound yields that the median of $O(\log \frac{n}{\delta})$ estimates will be an (ε, δ)-approximation for all (u, v) with $p_{uv} \geq p$.

The sets $\texttt{MinHash}_u$ for each user can be stored in an indexed array allowing constant time access. We maintain the k smallest hash values in a priority queue and update it for each newly performed action. We need h to be t-wise independent for $t = \mathcal{O}(\frac{1}{\varepsilon^2 p})$ and by [12] h can be represented in space $\mathcal{O}(\frac{1}{\varepsilon^2 p})$ and evaluated in time $\mathcal{O}(\log^2 \frac{1}{\varepsilon p})$. □

Algorithm 2: STRIP - Linear Memory

Compute Samples
Input: stream $\mathcal{S}$ of actions, threshold τ, a parameter k, t-wise independent hash function $h : A \to [0, 1]$
for $(u, a, t) \in \mathcal{S}$ **do**
  $\texttt{MinHash}_u$.update($(h(a), t), k$).

Single Estimate
Input: users u, v, a set of samples $\texttt{MinHash}_z$ for all users z
Let $\texttt{MinHash}^k_{u|v}$ be the k entries with the smallest hash value in $\texttt{MinHash}_u \cup \texttt{MinHash}_v$
return $\frac{\text{Prop}(\texttt{MinHash}^k_{u|v}, \tau)}{k}$

5.3 Sliding window model

We extend the previous algorithm to estimate the influence probabilities over sliding windows. The main idea is to guarantee that the required $\mathcal{O}(\frac{1}{\varepsilon^2 p})$ smallest hash values of actions performed within the window are kept with constant error probability. We achieve this by adjusting the exponential-histogram technique.

Theorem 5. *Let $G = (V, E)$ be a connected graph over n vertices and m edges, $\mathcal{S}$ be a stream of s actions and W a time-based sliding window of at most w actions. There is an algorithm returning an (ε, δ)-approximation of the influence probability p^W_{uv} of all edges with $p^W_{uv} \geq p$ in amortized time $\mathcal{O}((s \log^2(\frac{1}{\varepsilon p}) + \frac{m}{\varepsilon^2 p}) \log \frac{n}{\delta})$ and space $\mathcal{O}(\frac{n}{\varepsilon^2 p} \log^2(\frac{1}{\varepsilon p}) \log w \log \frac{n}{\delta})$ in one pass over $\mathcal{S}$.*

Proof. Assume for each i we maintain t buckets of width 2^i, $0 \leq i \leq \log \frac{w}{2t}$ for $t \geq \frac{1}{\varepsilon}$. Similarly to the proof of Theorem 2, we can show that storing $\mathcal{O}(\frac{1}{p\varepsilon} \log \frac{1}{\varepsilon p})$ values per bucket, we can compute with constant error probability $p_1 < 1/4$ the required smallest $k = \mathcal{O}(\frac{1}{\varepsilon^2 p})$ hash values.

The construction of exponential histograms implies that in the last bucket, i.e., the bucket keeping track of the oldest actions, we might have recorded k hash values of actions not performed within the window. Consider a directed edge (u, v) and assume the worst case that we have no record of $w/(2t)$ actions performed by u within the window and all of them have propagated from u to v. This happens when either for u or w we have not recorded hash values of actions performed within the window and propagating from u to v. Therefore, as shown in Theorem 4 we can obtain with error probability less than 1/2 an $(1 \pm \varepsilon)$-approximation of the quantity $\frac{s(A^w_{u2v}) - w/(2t)}{s(A_{u|v})}$ for $k = \mathcal{O}(\frac{1}{\varepsilon^2 p})$. This yields an additive approximation of $\varepsilon p_{uv} - \frac{w}{k s(A_{u|v})}$. From $s(A_{u|v}) \leq 2w$ and $p_{uv} \geq p$ we obtain that for $t \geq \frac{1}{p\varepsilon}$ we will have an $(1 \pm \varepsilon)$-multiplicative approximation with error probability $p_2 < 1/4$. Similar reasoning applies to the case when we have no record of $w/(2t)$ actions performed by u and no of them propagated to v.

Thus, with error probability less than 1/2 we have an $(1 \pm \varepsilon)$-approximation of the influence probability for a given arc. Running $\mathcal{O}(\log \frac{n}{\delta})$ copies in parallel and taking for each edge the median of the estimates, the approximation holds for all arcs with probability at least $1 - \delta$. □

Algorithm 3: STRIP - Sublinear Memory

ComputeSamplesInBuckets
Input: stream $\mathcal{S}$ of actions performed by users, int k, pairwise independent hash function $g : V \to [\ell]$, r-wise independent hash function $h : A \to [0, 1]$
for $(u, a, t) \in \mathcal{S}$ **do**
  $q = g(u)$
  $\texttt{MinHash}_q$.update($(u, h(a), t), k$).

Single Estimate
Input: users u, v, threshold τ, a set of samples $\{\texttt{MinHash}_q\}, q \in [\ell]$
$b_u = g(u), b_v = g(v)$
if $|\{(u, h(a), t_u) \in \texttt{MinHash}_{b_u}\}| \geq k/2$ **and** $|\{(v, h(a), t_v) \in \texttt{MinHash}_{b_v}\}| \geq k/2$ **then**
  Let $\texttt{MinHash}^k_u$ and $\texttt{MinHash}^k_v$ be the k triples with the smallest hash values for actions performed by u and v, respectively
  Let $\texttt{MinHash}^k_{u|v}$ be the k entries with the smallest hash value in $\texttt{MinHash}_u \cup \texttt{MinHash}_v$
  return $\frac{\text{Prop}(\texttt{MinHash}^k_{u|v}, \tau)}{k}$

6. SUBLINEAR SPACE

For the landmark model we combine the algorithm from Section 5.2 with hashing based algorithms for mining heavy hitters in data streams, e.g., [6]. The main idea in these algorithms is to distribute the heavy items to different bins by a suitably defined hash function. Then one shows that the contribution from non-heavy items in each bin is not significant and one can obtain high quality estimates for the heaviest items. In Algorithm 3 we apply the approach to active users but instead of estimating the activity, we are interested in obtaining a sample of the actions performed by active users. By recording the actions with the smallest hash values in each bin, we will show that with high probability a large fraction of those will be for actions performed by an active user.

Theorem 6. *Let $G = (V, E)$ be a connected graph over n vertices and m edges, and $\mathcal{S}$ be a stream of s actions. Let the user activity follow Zipfian distribution with parameter z and let $(u_1, u_2, \ldots, u_n)$ be the users sorted according to their*

activity. There exists a one-pass streaming algorithm computing an (ε, δ)-approximation of the influence probabilities of all edges (u_i, u_j) with $p_{u_i u_j} \geq p$ for $i, j \leq b$ in time T and space S such that:

- *if $z < 1$, then $S = \mathcal{O}(\frac{n^{1-z}b^z}{\varepsilon^2 p} \log \frac{b}{\delta})$ and $T = \mathcal{O}((s \log \frac{1}{\varepsilon p} + \frac{m}{\varepsilon^2 p}) \log \frac{b}{\delta})$.*
- *if $z > 1$, then $S = \mathcal{O}(\frac{b}{\varepsilon^2 p} \log \frac{b}{\delta})$ and $T = \mathcal{O}((s \log \frac{1}{\varepsilon p} + \frac{m}{\varepsilon^2 p}) \log \frac{b}{\delta})$.*

We can extend the algorithm to handle the sliding window model under the assumption that the user activity within the window adheres to Zipfian distribution. We adjust the exponential histograms approach such that each bucket records a certain number of bins recording the smallest hash values. As in the previous sections, this incurs an additional cost of $polylog(\frac{1}{\varepsilon p}, w)$ to the space complexity of the algorithm.

7. EXPERIMENTAL EVALUATION

The goal of our experiments is twofold. First, to show that the proposed approach is indeed applicable to learning influence probabilities in real social networks and yields results that confirm the theoretical analysis. Second, to compare the algorithms for the different memory models. Note that we did not try to optimize the space usage since it heavily depends on low-level technical details. For example, in our Java implementation we worked with data structures provided by the `java.util` package. However, a hash table then automatically wraps primitive data types as objects, which considerably increases the space usage. While such details can greatly influence the performance, they are beyond the scope of the present work.

Details on the implementation. We used tabulation hashing [5] to implement the hash function. (See, e.g., [20] for details.) The approach is very efficient and each hash function can be stored in fast cache and evaluated in constant time. The scheme yields only 3-wise independence but recently Chernoff-like concentration on the estimates provided by algorithms using tabulation hashing with a *single* hash function were shown [20]. Therefore, we worked with only one hash function instead of taking the median of several estimates provided by different functions.

Experimental settings. The algorithms were implemented in Java and experiments performed on a Windows 7 machine with 3.30 GHz clocked processor and 4 GB RAM. We experiment on two real-world datasets. For both datasets we extend the available stream of events, by creating synthetic data in a smart way: the synthetic stream is generated by an instance of the Independent Cascade model that fits the available real data, as explained next.

The first dataset is obtained by crawling the public timeline of Twitter and tracking the propagation of hashtags across the network. The second dataset has been crawled from Flixster [1], one of the main web communities which allows to share ratings on movies and to meet other users with similar tastes. The propagation log records the time at which a user rated a given movie. An item propagates from v to u, if u rates the item shortly after the rating by v. The datasets basic statistics are reported in Table 1.

The user activity is quite skewed: the maximum number of actions performed by a user in Twitter is 358 and in Flixster 11,542; in Twitter the 1,000 most active users perform 68,191 actions, and in Flixster the skew is even more significant with 1,000 users performing 1,643,686 actions.

[1] `http://www.cs.sfu.ca/sja25/personal/datasets/`

	$\|V\|$	E	Δ	actions
Twitter	26,488	1,636,462	6,883	580,141
Flixster	29,357	425,228	585	6,529,008

Table 1: Characteristics of the real-world datasets.

From the real datasets we create a larger stream of actions as follows. For a suitably chosen propagation threshold τ we compute the influence probabilities among edges and also compute the "starting" probability for each user to initiate a given action, i.e., how probable is that a given user performs an action without being influenced by its neighbors. Then we sequentially create new items. For a given new item, from the starting probabilities for each user we toss a biased coin and decide whether the user will perform the action. If performed, the action is then propagated to its neighbors according to the precomputed influence probability within time r, where r is a random number in $(0, \tau]$. The average influence probability in Twitter is 0.0099 and the number of edges with influence probability at least 0.05 is 92,813. The numbers for Flixster are 0.0202 and 51,963, respectively.

For 100,000 items propagated through the Twitter network we obtain a stream of about 36 million actions. For the Flixster network we propagate 1,000,000 items obtaining a stream of about 27 million actions. The synthetic datasets exhibit similar characteristics to the original data with skewed activity among users and, not surprisingly, similar distribution of the influence probabilities.

For a directed edge (u, v), we denote the approximation of the influence probability p_{uv} by $\tilde{p}_{uv}$. We evaluate the quality of the estimates of all edges (u, v) s.t. $p_{uv} \geq 0.05$ with respect to average relative error, Pearson correlation coefficient and Spearman rank correlation [3].

Evaluation. Table 3 shows the quality of the estimates for varying number of samples. As expected, the best estimates are obtained for Algorithm 1, the superlinear space model, since the number of propagations for any (u, v) is computed exactly and we only estimate the number of different actions performed by u or v. The left-most plot in Figure 1 confirms that the quality of the approximation does not depend on the influence probability. Due to the sparsity of the graph structure of the two considered networks, the space complexity of Algorithm 1 is comparable to the space-usage by Algorithm 2. However, the running time is more than an order of magnitude larger and this is due to the fact that for each incoming triple (u, a, t_u) we explicitly need to check whether v has performed a within τ time units of t_u for all arcs (v, u). Instead, in Algorithm 2 each incoming triple is processed in constant amortized time. The exact numbers are in Table 2, where the space is the number of stored samples and the time is given in seconds. For the sublinear model we estimated the influence probabilities among the 1,000 most active users and worked with 3,000 bins. For a given number of samples x we then stored in each bin the $3x$ smallest hash values of users hashed to the bin. The plots in Figure 1 confirm that despite of working with a single hash function there are no outliers in the estimates.

In Tables 4 and 5 we evaluated the quality of estimates for time-based sliding windows. We choose a time threshold for the window such that the number of actions in the windows is approximately 10% and 20% of the stream, respectively. In order to obtain the smallest k hash values

		Superlinear memory		Linear memory		Sublinear memory	
	Samples	Space	Time	Space	Time	Space	Time
Twitter	100	1,538,993	1401.27	1,639,720	44,45	893,344	39,48
	150	2,342,512	1389.76	2,338,055	42,97	1,337,046	41,05
	200	3,081,681	1387.96	3,005,698	45,17	1,777,084	43,52
	250	3,639,120	1393.65	3,646,400	47,73	2,222,563	44,79
	300	4,212,704	1395.02	4,261,574	51,32	2,641,752	45,69
	350	4,799,429	1393.86	4,854,004	54,05	3,089,974	47,65
	400	5,478,576	1397.03	5,425,154	55,86	3,514,610	50,36
	450	5,925,660	1396.58	5,977,632	56,17	3,950,878	52,18
	500	6,498,569	1397.16	6,514,259	56,44	4,405,461	54,38
Flixster	100	2,031,217	137.58	1,993,107	29.63	897,042	28.73
	150	2,652,329	138.37	2,732,788	28,79	1,343,034	30.03
	200	3,357,899	137.05	3,400,342	30,07	1,779,307	31,24
	250	4,020,016	138.18	4,013,011	32,36	2,220,812	34,12
	300	4,701,442	138.59	4,583,334	33,66	2,650,249	35.51
	350	5,098,353	139.06	5,118,537	34,45	3,085,930	38,62
	400	5,559,797	138.98	5,625,570	35,03	3,505,002	39,22
	450	6,092,310	139.78	6,107,170	36,72	3,926,346	39,95
	500	6,519,207	140.02	6,565,227	38,42	4,339,582	41,11

Table 2: Complexty of the algorithms. The running time is given in seconds and the space is the number of samples stored in memory.

		Superlinear memory			Linear memory			Sublinear memory		
	Samples	Avg Error	Pearson	Spearman	Avg Error	Pearson	Spearman	Avg Error	Pearson	Spearman
Twitter	100	0.0573	0.9908	0.9931	0.2682	0.8495	0.8251	0.2327	0.5675	0.546
	150	0.0502	0.9915	0.9939	0.2178	0.8919	0.8727	0.1887	0.6584	0.6261
	200	0.0461	0.9922	0.9948	0.1881	0.9155	0.9022	0.1644	0.7024	0.6727
	250	0.0402	0.9937	0.9960	0.1678	0.9295	0.9175	0.1474	0.7374	0.7112
	300	0.0372	0.9951	0.9964	0.1519	0.9422	0.9304	0.1355	0.7571	0.7286
	350	0.0301	0.9969	0.9971	0.1405	0.9492	0.9408	0.1232	0.7900	0.7615
	400	0.0283	0.9973	0.9982	0.1304	0.9552	0.9472	0.1158	0.8043	0.7745
	450	0.0226	0.9974	0.9989	0.1226	0.9607	0.9529	0.1068	0.8318	0.8031
	500	0.0243	0.9991	0.9994	0.1160	0.9639	0.9583	0.1016	0.8465	0.8229
Flixster	100	0.0637	0.9612	0.9813	0.2761	0.6913	0.8316	0.3215	0.7554	0.6154
	150	0.0585	0.9699	0.9852	0.2242	0.7511	0.8795	0.268	0.8202	0.6792
	200	0.0493	0.9761	0.9888	0.1891	0.7906	0.9076	0.223	0.867	0.7437
	250	0.0413	0.9807	0.9911	0.1669	0.8215	0.9238	0.2055	0.877	0.7539
	300	0.0364	0.9851	0.9942	0.1499	0.8382	0.9366	0.1866	0.898	0.7851
	350	0.0320	0.9866	0.9950	0.1369	0.8581	0.9448	0.1723	0.9115	0.801
	400	0.0291	0.9889	0.9958	0.1278	0.8706	0.9529	0.1625	0.9204	0.8198
	450	0.0275	0.9901	0.9965	0.1184	0.8817	0.9584	0.1494	0.9261	0.8321
	500	0.0264	0.9922	0.9971	0.1109	0.8904	0.9625	0.1482	0.9297	0.8361

Table 3: Quality of the estimates for the landmark model for various number of samples.

		Superlinear memory			Linear memory			Sublinear memory		
	Samples	Avg Error	Pearson	Spearman	Avg Error	Pearson	Spearman	Avg Error	Pearson	Spearman
Twitter	50	0.0685	0.9951	0.9967	0.252	0.8345	0.8151	0.266	0.4099	0.4289
	100	0.0432	0.9961	0.9571	0.2082	0.879	0.8621	0.1841	0.5056	0.5476
	150	0.0374	0.9983	0.9986	0.1793	0.9189	0.904	0.1539	0.6045	0.6461
	200	0.0335	0.9969	0.9986	0.1582	0.9286	0.9165	0.1323	0.6747	0.7112
	250	0.0257	0.9973	0.9985	0.1423	0.9412	0.9304	0.1373	0.6511	0.703
	300	0.0329	0.9993	0.9992	0.1349	0.9492	0.938	0.1315	0.7571	0.7286
Flixster	50	0.1281	0.9036	0.9617	0.2481	0.7563	0.782	0.4532	0.5305	0.6517
	100	0.0555	0.97	0.9897	0.202	0.7812	0.8362	0.335	0.6221	0.7506
	150	0.0439	0.9791	0.993	0.1901	0.8206	0.8512	0.267	0.6868	0.828
	200	0.0357	0.9861	0.9956	0.162	0.8611	0.8894	0.2211	0.7521	0.8737
	250	0.0304	0.989	0.9965	0.1519	0.8821	0.9005	0.201	0.7765	0.8897
	300	0.0285	0.9902	0.9969	0.1465	0.9152	0.9246	0.1762	0.7983	0.912

Table 4: Quality of estimates for a sliding window of length approximately 10% of the stream.

		Superlinear memory			Linear memory			Sublinear memory		
	Samples	Avg Error	Pearson	Spearman	Avg Error	Pearson	Spearman	Avg Error	Pearson	Spearman
Twitter	50	0.1612	0.9565	0.9675	0.2609	0.8364	0.8562	0.2603	0.4952	0.4785
	100	0.1282	0.9861	0.9841	0.2345	0.875	0.891	0.1931	0.5562	0.5748
	150	0.0434	0.9883	0.9944	0.1766	0.9104	0.9248	0.1475	0.6674	0.695
	200	0.0305	0.9959	0.9968	0.1491	0.9212	0.9324	0.1278	0.6989	0.7353
	250	0.038	0.9976	0.9978	0.1385	0.9432	0.9514	0.1113	0.7514	0.7855
	300	0.0195	0.9973	0.9985	0.1215	0.9502	0.9667	0.1017	0.7834	0.8127
Flixster	50	0.1033	0.9034	0.9722	0.2434	0.7292	0.8735	0.4514	0.5366	0.6696
	100	0.0528	0.9569	0.9906	0.1931	0.8183	0.8975	0.3229	0.6416	0.7761
	150	0.0325	0.9751	0.9953	0.1602	0.8243	0.9362	0.2595	0.7097	0.8346
	200	0.0291	0.9806	0.9962	0.1469	0.8508	0.9538	0.2266	0.7469	0.8623
	250	0.0268	0.9837	0.9971	0.122	0.8777	0.9606	0.206	0.76	0.8853
	300	0.0175	0.9911	0.9985	0.1009	0.893	0.9648	0.182	0.7894	0.922

Table 5: Quality of estimates for a sliding window of length approximately 20% of the stream.

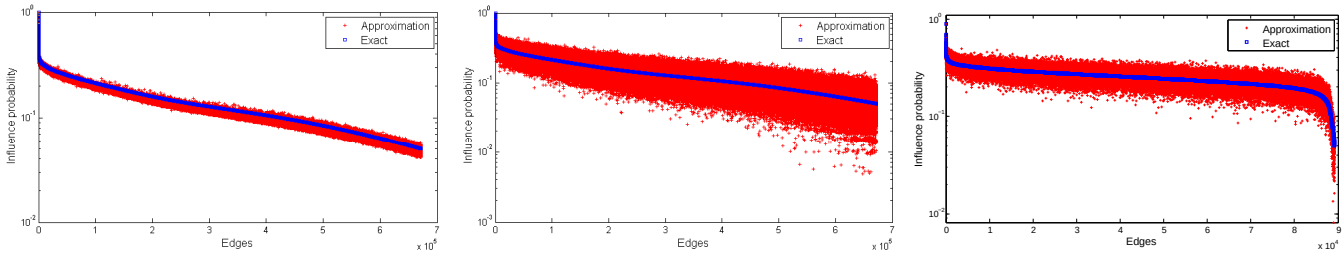

Figure 1: Visualization of the estimates for the Twitter dataset for the three memory models.

overall, we observed that tabulation hashing's randomness allows us to store only the smallest $\mathcal{O}(k \log k/t)$ hash values in each of the t buckets of given width. The processing of the stream was stopped at a random point and we evaluated the quality of the estimates. Due to lack of space we do not include the exact time and space complexities but we observe that for the smaller window the space-savings are small, using between 50% and 85% of the total window size for a varying number of samples. For the larger window, better space savings were observed using between 35% and 70% of the window size, clearly indicating that our approach is advantageous to storing the whole window only for larger windows. For the sublinear space model we concentrated only on the 500 top users and even for the smaller window were able to achieve relatively good space savings, varying between 18% and 40% of the window.

In a summary, we see reasonably good estimates on the influence probabilities by storing just a few hundreds hash values. For larger streams the advantages of the approach become more significant. Also, while the superlinear memory algorithm yields better estimates, this comes at the price of a much worse processing time. For denser social networks this drawback will be even more pronounced.

8. CONCLUSION AND FUTURE WORK

Microblogging platforms as `twitter` are becoming large real-time generators of social data-streams. In this paper, we presented STRIP, a suite of streaming methods for computing the influence strength along each link of a social network. To the best of our knowledge, these are the first streaming methods that compute influence probabilities. The STRIP methods builds upon a wise use of probabilistic approximations, min-wise independent hashing functions, and streaming sliding windows. These methods works in several scenarios, depending on the available memory and whether we are interested in the whole history of the stream or only in more recently performed actions.

In our future investigation, we plan to extend the present work in two directions. First, we plan to use adaptive size windows, so that the data analyst does not need to decide *a priori* what is the optimal window. Second, we plan to implement the STRIP suite of methods using distributed stream systems, to be able to process social data-streams in a distributed fashion.

Acknowledgements. This work done while the first author was visiting Yahoo! Research, Barcelona and supported by the Danish National Research Council under the Sapere Aude program. We would like to thank Rasmus Pagh for valuable suggestions about the analysis of the algorithm.

9. REFERENCES

[1] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM'02*.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[3] C. Brase and C. Brase. *Understandable Statistics: Concepts and Methods*. Brooks/Cole, 2011.

[4] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[5] L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[7] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD'10*.

[8] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Trans. Knowl. Data Eng.*, 13(1):64–78, 2001.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

[10] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD'01*.

[11] J. Feigenbaum, S. Kannan, A. McGregor, and J. Zhang. On graph problems in a semi-streaming model. In *ICALP*, 2004.

[12] G. Feigenblat, E. Porat, and A. Shiftan. Exponential time improvement for min-wise based algorithms. *Inf. Comput.*, 209(4):737–747, 2011.

[13] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM'10*.

[14] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1):73–84, 2011.

[15] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[16] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD'03*.

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD'05*.

[18] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD'07*.

[19] R. Motwani and P. Raghavan. *Randomized Algorithms*. CRC Press, 1997.

[20] M. Pătraşcu and M. Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14, 2012.

[21] K. Saito, R. Nakano, and M. Kimura. Prediction of information diffusion probabilities for independent cascade model. In *KES'08*.

10. APPENDIX

PROOF. (of Theorem 3) We reduce the BIT-VECTOR DISJOINTNESS problem to detecting edges with high influence probability in streamed graphs. BIT-VECTOR DISJOINTNESS is the following problem. There are two bit strings $A, B \in \{0,1\}^n$ indexed from 1 to n. Alice holds A, Bob holds B and they want to decide whether there exists an $1 \leq i \leq n$ such that the ith bit is set to 1 in both A and B. Alice and Bob want to communicate to each other a minimum number of bits. Any randomized protocol for the problem achieving error probability of less than 1/2 needs $\Omega(n)$ bits [15].

The reduction is as follows. (We refer to Figure 2 for an example.) Let $V = \{v_1, \ldots, v_n\}$ be a set of vertices which will correspond to the bit position in A and B. Further, let $Z^A = \{z_j^A\}$ and $Z^B = \{z_j^B\}$, $1 \leq j \leq d-1$, be two sets of of $d-1$ vertices each. We first read Alice's string bit by bit and if the ith bit is set to 0 we create $d-1$ edges (v_i, z_j^A). Otherwise, if $A[i] = 1$ we create the edge (v_i, o_i^A) for a vertex o_i not contained in $Z^A \cup Z^B$. The same is then repeated for the bits in B with the difference that for 0-bits we add the edges $\{v_i, z_j^B\}$ and for 1-bits – the edge (v_i, o_j^B). At the end we connect v_i and v_{i+1} for $1 \leq i \leq n-1$.

We then let each v_i perform $2(d-1)$ different actions. Let $c = |\{u \in N(v_i), u \notin V\}|$, i.e., the number v_i's neighbors not contained in V. Assuming a partial order on v_i's neighbors, we propagate the ith action to the neighbor in position $i \bmod c$.

We see that if there exists an i such that in A and B the ith bit is set to 1, then there exists an edge, namely (v_i, o_i^A) and (v_i, o_i^B), with a influence probability of 1/2. On the other hand, if at most one of the ith bits in A and B is set to 1, then u_i performs $2(d-1)$ actions and most 2 actions are propagated to any of its neighbors.

The above construction is deterministic, therefore we can feed an algorithm for detecting edges with high influence probability with the same input more than once. The lower bound on the communication complexity of randomized algorithms for the BIT-VECTOR-DISJOINTNESS problem is $\Omega(n)$ bits. □

Note that the number of edges in the constructed graph is of the order $O(dn)$, and for constant d the lower bound of $\Omega(n)$ for BIT-VECTOR DISJOINTNESS implies also a lower bound of $\Omega(m)$ bits for learning influence probabilities in a streaming setting. This however does not rule out the existence of an algorithm with space complexity $o(m)$ for a given m but implies that we cannot obtain an $o(m)$-space algorithm for sparse instances, which is the case for other problems.

PROOF. (of Theorem 6) Let the number of bins be $\ell \geq 8b$. Consider a given active user u_i, $i \in [b]$, and assume $g(u_i) = q$, i.e., g hashes u_i to some bin $q \in [\ell]$. In the following we will abuse notation and denote by $g(u_i)$ the bin u_i is hashed to by g.

Since g is pairwise independent, for each other active user u_j it holds $\Pr[g(u_j) = q] = \frac{1}{8b}$. Thus, by Markov's inequality with probability at most 1/8 there is another active user hashed to the same bin as u_i.

Let $w = \sum_{i=b+1}^{n} \frac{s}{\zeta(z) i^z}$, i.e., the total weight of user activity of non-active users. We expect $\frac{w}{8b}$ actions by non-active users to be hashed to $g(u_i)$. We analyze the required number ℓ of bins in order to guarantee that with probability at

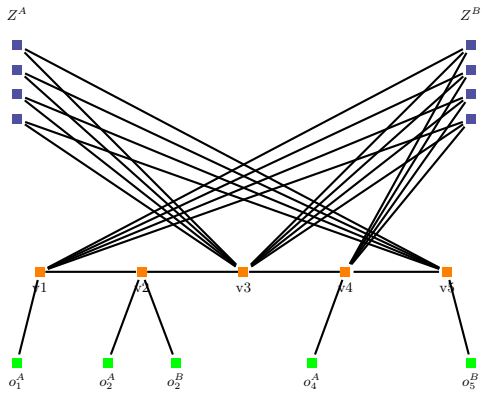

Figure 2: An example of the construction for proving the lower bound for bit-vectors consisting of 6 bits. The edges (v_2, o_2^A) and (v_2, o_2^B) have each influence probability 1/2 and all other edges have influence probability at most 1/5.

least 3/4 non-active users with a total number of actions not more than $\frac{s}{2\zeta(z)b^z}$ will land in $g(u_i)$.

- For $z < 1$ we have $w = \sum_{i=b+1}^{n} i^{-z} \leq 2n^{1-z}$. Thus, for $\ell = 8n^{1-z}b^z$ buckets, again by Markov's inequality, we bound the probability that the total activity of non-active users will exceed $\frac{s}{2\zeta(z)b^z}$ by 1/4. It also holds $n^{1-z}b^z \geq b$ for $b \leq n$ and $0 < z < 1$.
- For $z > 1$ we have $w = \sum_{i=b+1}^{n} i^{-z} \leq 2b^{1-z}$. Thus, for $\ell = 8b$ we bound the probability for more than $\frac{s}{2\zeta(z)b^z}$ actions by non-active users to 1/4.

By the union bound we then conclude that with probability at least 5/8 an active user u_i will be the only active user in $g(u_i)$ and the number of actions performed by him/her will be at least twice the actions performed by other users and hashed to $g(u_i)$. We need $k = \mathcal{O}(\frac{1}{\varepsilon^2 p})$ minimum hash values. Assume in each bin we record the $2k$ minimum hash values. Applying the same reasoning as in the proof of Theorem 4, we obtain that for h being $(\alpha, 2k)$-wise independent, with error probability p_1 the k actions performed by u_i with the smallest hash values are recorded. Consider now an edge (u_i, u_j), $1 \leq i, j \leq b$. As shown in the proof of Theorem 4, with error probability p_2 the k minimum hash values for u_i and u_j yield an $(1 \pm \varepsilon)$ of $p_{u_i u_j}$. One can choose a sufficiently small constant α such that $p_1 + p_2 < 1/2$, thus the median of $\mathcal{O}(\log \frac{1}{\delta})$ approximations is an (ε, δ)-approximation.

The time and space complexity follow directly from the above discussion and the proof of Theorem 4. □