Information Propagation in Interaction Networks

Rohit Kumar
Department of Computer and Decision Engineering
Université Libre de Bruxelles, Belgium
Department of Service and Information System Engineering
Universitat Politécnica de Catalunya, Spain
rohit.kumar@ulb.ac.be

Toon Calders
Department of Computer and Decision Engineering
Université Libre de Bruxelles, Belgium
Department of Mathematics and Computer Science
Universiteit Antwerpen, Belgium
Toon.Calders@uantwerpen.be

ABSTRACT

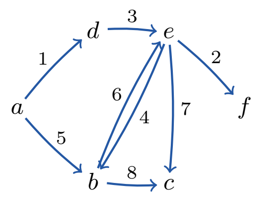
We study the potential flow of information in interaction networks, that is, networks in which the interactions between the nodes are being recorded. The central notion in our study is that of an *information channel*. An information channel is a sequence of interactions between nodes forming a path in the network which respects the time order. As such, an information channel represents a potential way information could have flown in the interaction network. We propose algorithms to estimate information channels of limited time span from every node to other nodes in the network. We present one exact and one more efficient approximate algorithm. Both algorithms are one-pass algorithms. The approximation algorithm is based on an adaptation of the HyperLogLog sketch, which allows easily combining the sketches of individual nodes in order to get estimates of how many unique nodes can be reached from groups of nodes as well. We show how the results of our algorithm can be used to build efficient *influence oracles* for solving the *Influence maximization problem* which deals with finding top k seed nodes such that the information spread from these nodes is maximized. Experiments show that the use of information channels is an interesting data-driven and model-independent way to find top k influential nodes in interaction networks.

Keywords

Influence Maximization, Influence estimation, Information flow mining

1. INTRODUCTION

In this paper, we study information propagation by identifying potential "information channels" based on interactions in a dynamic network. Studying the propagation of information through a network is a fundamental and well-studied problem. Most of the works in this area, however, studied the information propagation problem in static networks or graphs only. Nevertheless, with the recent advancement in data storage and processing,



(a)

src.	dst.	time
b	c	8
e	c	7
b	e	6
.	..	..
e	f	2
a	d	1

(b)

Figure 1: (a) An example Interaction graph. (b) The interaction in reverse order of time.

it is becoming increasingly interesting to store and analyze not only the connections in a network but the complete set of interactions as well. In many networks not only the connections between the nodes in the network are important, but also and foremost, how the connected nodes interact with each other. Examples of such networks include email networks, in which not only the fact that two users are connected because they once exchanged emails is important, but also how often and with whom they interact. Another example is that of social networks where people become friends once, but may interact many times afterward, intensify their interactions over time, or completely stop interacting. The static network of interactions does not take these differences into account, even though these interactions are very informative for how information spreads. To illustrate the importance of taking the interactions into account, Kempe et al. [12] showed how the temporal aspects of networks affect the properties of the graph.

Figure 1a gives an example of a toy interaction network. As can be seen, an interaction network is abstracted as a sequence of timestamped edges. A central notion in our study is that of an *information channel*; that is, a path consisting of edges that are increasing in time. For instance, in Figure 1a, there is an information channel from a to e, but not from a to f. This notion of an information channel is not new, and was already studied under the name *time-respecting path* [12] and is a special case of *temporal paths* [26].In contrast to earlier work on information channels we additionally impose a constraint on the total duration of the information channel, thus reflecting the fact that in influence propagation the relevance of the message being propagated may deteriorate over time. To the best of our knowledge, our paper is the first one to study the notion of temporal paths with time constraints in influence propagation on interaction networks.

We propose a method to identify the most influential nodes in the network based on how many other nodes they could potentially reach through an information channel of limited timespan. As such, the information channels form an implicit propagation model learned from data. Most of the related work in the area of information propagation in interaction or dynamic networks uses probabilistic models like the independent cascade(IC) model or the Linear Threshold(LT) model, and tries to learn the influence probabilities that are assumed to be given by these models [13, 4, 3, 6]. Another set of recent work focuses on deriving the hidden diffusion network by studying the cascade information of actions [10, 11] or cascade of infection times [8, 24]. These paper, however, use a very different model of interactions. For example, the work by Goyal et al. [10, 11], every time an activity of a node a is repeated within a certain time span by a node b that is connected to a in the social graph, this is recorded as an interaction. Each user can execute each activity only once, and the strength of influence of one user over the other is expressed as the number of different activities that are repeated. While this model is very natural for certain social network settings, we believe that our model is much more natural for networks in which messages are exchanged, such as for instance email networks because activities such as sending an email can be executed repeatedly and already include the interaction in itself. Furthermore, [11] is not based on information channels, but on the notion of credit-distribution, and [10] does not include the time-respecting constraint for paths.

One of the key differentiators of the techniques introduced here and earlier work is that next to an exact algorithm, we also propose an efficient one-pass algorithm for building an approximate influence oracle that can be used to identify top-k maximal influencers. Our algorithm is based on the same notion as shown in so-called *sliding window HyperLogLog sketch* [15] leading to an efficient, yet approximate solution. Experiments on various interaction networks with our algorithm show the accuracy and scalability of our approximate algorithm, as well as how it outperforms algorithms that only take into account the static graph formed by the connected nodes.

The contribution of this paper are as follows.

- Based on the notion of an *Information Channel*, we introduce the *Influence Reachability Set* of a node in a interaction network.
- We propose an exact but memory inefficient algorithm which calculates the *Influence Reachability Set* of every node in the network in one pass over the list of interactions.
- Next to the exact algorithm, an approximate sketch-based extension is made using a *versioned* HyperLogLog sketch.
- With the influence reachability sets of the nodes in our interaction network, we identify top-k influencers in a model-independent way.
- We propose a new Time Constrained Information Cascade Model for interaction networks derived from the Independent Cascade Model for static networks.
- We present the results of extensive experiments on six real world interaction network datasets and demonstrate the effectiveness of the time window based influence spread maximization over static graph based influence maximization.

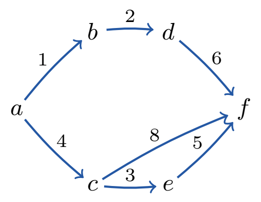

$$\begin{tikzcd} & b \arrow[r, "2"] & d \arrow[dr, "6"] & \\ a \arrow[ur, "1"] \arrow[dr, "4"] & & & f \\ & c \arrow[r, "3"] \arrow[urr, "8"] & e \arrow[ur, "5"] & \end{tikzcd}$$

Figure 2: Interaction network example with multiple information channels between node c and f

2. PRELIMINARIES

Let V be a set of nodes. An *interaction* between nodes from V is defined as a triplet (u, v, t), where $u, v \in V$, and t is a natural number representing a time stamp. The interaction (u, v, t) indicates that node u interacted with node v at time t. Interactions are directed and could denote, for instance, the sending of a message. For a directed edge $u \rightarrow v$, u is the source node and v is the destination node. An interaction network $G(V, \mathcal{E})$ is a set of nodes V, together with a set $\mathcal{E}$ of interactions. We assume that every interaction has a different time stamp. We will use $n = |V|$ to denote the number of nodes in the interaction network, and $m = |\mathcal{E}|$ to denote the total number of interactions.

Time Constrained Information Cascade Model: For interaction networks, influence models such as the Independent Cascade Model or Linear Threshold Model no longer suffice as they do not take the temporal aspect into account and are meant for static networks. To address this shortcoming, we introduce a new model of Information Cascade for Interaction networks. The *Time Constrained Information Cascade Model* (TCIC) is a variation of the famous *Independent Cascade Model*. This model forms the basis of our comparison with other baselines SKIM [6], PageRank and High Degree. We say a node is *infected* if it is influenced. For a given set of seed nodes we start by infecting the seed nodes at their first interaction in the network and then start to spread influence to their neighbors with a fixed probability. The influence spread is constrained by the time window(ω) specified; i.e, once a seed node is infected at time stamp t it can spread the infection to another node via a temporal path only if the interaction on that path happens between time t and $t + \omega$. For sake of simplicity we use a fixed infection probability in our algorithms to simulate the spread nevertheless node specific probabilities or random probabilities could easily be used as well. In Algorithm 1 we present the algorithm for the TCIC model.

In order to Find highly influential nodes under the TCIC model we introduce the notion of Information Channel.

Definition 1. (Information Channel) *Information Channel ic* between nodes u and v in an interaction network $G(V, \mathcal{E})$, is defined as a series of time increasing interactions from $\mathcal{E}$ satisfying the following conditions: $ic = (u, n_1, t_1), (n_1, n_2, t_2), ...(n_k, v, t_k)$ where $t_1 < t_2 < .. < t_k$. The *duration* of the information channel ic is $dur(ic) := t_k - t_1 + 1$ and the *end time* of the information channel ic is $end(ic) := t_k$. We denote the set of all information channels between u and v as $IC(u, v)$, and the set of all

Algorithm 1 Simulation with a given seed set and window

```
Input: $G(V,E)$ the interaction graph given as a time-ordered list $\ell_G$ of $(u,v,t)$, $\omega$, and $S$ the seed set. $p$ is the probability of infection spread on interaction.
Output: Number of nodes influenced by the seed.
Initially all nodes are inactive and for all activateTime is set to -1.
for all $(u,v,t) \in \ell_G$ do
    if $u \in S$ then
        u.isActive=true
        u.activateTime=t
    end if
    if u.isActive & $(t - u.activateTime) \leq \omega$ then
        With probability $p$
        v.isActive=true
        if u.activateTime > v.activateTime then
            v.activateTime=u.activateTime
        end if
    end if
end for
Return: Count of nodes for which isActive is true.
```

information channels of duration ω or less as $IC_\omega(u,v)$.

Notice that there can exist multiple information channels between two nodes u and v. For example, in Fig 2 there are 2 information channels from a to f. The intuition of the information channel notion is that node u could only have sent information to node v if there exists a time respecting series of interactions connecting these two nodes. Therefore, nodes that can reach many other nodes through information channels are more likely to influence other nodes than nodes that have information channels to only few nodes. This notion is captured by the *influence reachability set.*

Definition 2. (Influence reachability set) The *Influence reachability set (IRS)* $\sigma(u)$ of a node u in a network $G(V,\mathcal{E})$ is defined as the set of all the nodes to which u has an information channel:

$$\sigma(u) \;:=\; \{v \in V \mid IC(u,v) \neq \emptyset\} \;.$$

Similarly, the influence set for a given maximal duration ω is defined as

$$\sigma_\omega(u) = \{v \in V \mid \exists ic \in IC(u,v) : dur(ic) \leq \omega\} \;.$$

The *IRS* of a node may change depending on the maximal duration ω. For example, in Figure 2 $\sigma_3(a) = \{b,c,d\}$ and $\sigma_5(a) = \{b,c,d,f\}$. This is quite intuitive because as the maximal duration increases, longer paths become valid, hence increasing the size of the influence reachability set. Once we have the *IRS* for all nodes in a interaction network for a given window we can efficiently answer many interesting queries, such as finding top k influential nodes. Formally, the algorithms we will show in the next section solve the following problem:

Definition 3. (IRS-based Oracle Problem) Given an interaction network $G(V,\mathcal{E})$, and a duration threshold ω, construct a data structure that allows to efficiently answer the following type of queries: *given a set of nodes $V' \subseteq V$, what is the cardinality of the combined influence reachability sets of the nodes in V'; that is:* $\left|\bigcup_{v\in V'} \sigma_\omega(v)\right|$.

First we will present an exact but memory inefficient solution that will maintain the sets $\sigma_\omega(v)$ for all nodes v. Clearly this data structure will allow to get the exact cardinality of the exact influence reachability sets, by taking the unions of the individual influence reachability sets and discarding duplicate elements. The approximate algorithm on it's turn will maintain a much more memory efficient sketch of the sets $\sigma_\omega(v)$ that allows to take unions and estimate cardinalities.

3. SOLUTION FRAMEWORK

In this section, we present an algorithm to compute the IRS for all nodes in an interaction network in one pass over all interactions. In the following all definitions assume that an interaction network $G(V,\mathcal{E})$ and a threshold ω have been given. We furthermore assume that the edges are ordered by time stamp, and will iterate over the interactions in *reverse order* of time stamp. As such, our algorithm is a one-pass algorithm, as it treats every interaction exactly once and, as we will see, the time spent per processed interaction is very low. It is not a streaming algorithm because it can not process interactions as they arrive. The reverse processing order of the edges is essential in our algorithm, because of the following observation.

Lemma 1. Let $G(V,\mathcal{E})$ be an interaction network, and let (u,v,t) be an interaction with a time stamp before any time stamp in $\mathcal{E}$; i.e., for all interactions $(u',v',t') \in \mathcal{E}$, $t' > t$. $G'(V,\mathcal{E} \cup \{(u,v,t)\})$ denotes the interaction network that is obtained by adding interaction (u,v,t) to G. Then, for all $w \in V \setminus \{u\}$, $IRS_\omega(w)$ is equal in G and G'.

PROOF. Suppose that $IRS_\omega(w)$ changes by adding (u,v,t) to $\mathcal{E}$. This means that there must exist an information channel ic from w to another node in G' that did not yet exist in G. This information channel hence necessarily contains the interaction (u,v,t). As t was the earliest time in the interaction network G', (u,v,t) has to be the first interaction in this information channel. Therefore w must be u and thus $w \notin V \setminus \{u\}$. □

This straightforward observation logically leads to the strategy of reversely scanning the list of interactions. Every time a new interaction (u,v,t) is added, only the IRS of the source node u needs to be updated. Notice that there is no symmetric definition for the forward scan of a list of interactions; if a new interaction arrives with a time stamp later than any other time stamp in the interaction network, potentially the IRS of every node in the network changes, leading to an unpredictable and potentially unacceptable update time per interaction.

In order to exploit the observation of Lemma 1, we keep a summary of the interactions processed so far.

Definition 4. (IRS Summary) For each pair $u,v \in V$, such that $IC_\omega(u,v) \neq \emptyset$, $\lambda(u,v)$ is defined as the end time of the earliest information channel of length ω or less from u to v. That is:

$$\lambda(u,v) \;:=\; \min(\{end(ic) \mid ic \in IC_\omega(u,v)\})$$

The IRS summary $\varphi_\omega(u)$ is now defined as follows:

$$\varphi_\omega(u) = \{(v, \lambda(u,v)) \mid v \in IRS_\omega(u)\} \;.$$

That is, we will be keeping for every node u the list of all other nodes that are reachable by an information channel of duration at most ω. Furthermore, for every such reachable node v, we keep the earliest time it can be reached from u by an information channel. The IRS of a node u can easily be computed from $\varphi_\omega(u)$ as $\sigma_\omega(u) = \{v \mid \exists t : (v,t) \in \varphi(u)\}$. On the other hand, the information stored in the summary consisting of $\varphi(u)$ for every u is sufficient to efficiently update it whenever we process the next edge in the reverse order as we shall see.

Example 1. In Figure 2, $\varphi_3(a) = \{(b,1),(d,2),(c,4)\}$ and $\varphi_3(c) = \{(f,5),(e,3)\}$. There are 2 information channels between c and f, one with $dur(ic) = 1$ and $end(ic) = 8$ and another with $dur(ic) = 3$ and $end(ic) = 5$ and hence $\lambda(c,f) = 5$.

3.1 The Exact algorithm

We illustrate our algorithm using the running example in Figure 1a. Table 1b shows all the interactions for the graph reverse ordered by time stamp. Recall that we process the edges in time decreasing order. The algorithm is detailed in Algorithm 2. First, we initialize all $\varphi(u)$ to the empty set. Then, whenever we process an interaction (u,v,t), we know from Lemma 1 that only the summary $\varphi(u)$ may change. The following lemma explains how the summary $\varphi(u)$ changes:

Lemma 2. Let $G(V,\mathcal{E})$ be an interaction network, and let (u,v,t) be an interaction with a time stamp before any time stamp in $\mathcal{E}$; i.e., for all interactions $(u',v',t') \in \mathcal{E}$, $t' > t$. $G'(V, \mathcal{E} \cup \{(u,v,t)\})$ denotes the interaction network that is obtained by adding the interaction (u,v,t) to G. Let $\varphi'(u)$ denote the summary of u in G' and $\varphi(u)$ that in G. Then, $\varphi'(u) = \downarrow (\{(v,t)\} \cup \varphi(u) \cup \{(z,t') \in \varphi(v) \mid t' - t + 1 \leq \omega\})$, where $\downarrow(A)$ denotes $A \setminus \{(v,t) \in A \mid \exists (v,t') \in A : t' < t\}$.

PROOF. Let ic be an information channel of duration maximally ω from u to z in G' that minimizes $end(ic)$. Then there are three options: (1) ic is the information channel from u to v formed by the single interaction (u,v,t) that was added. The end time of this information channel is t. (2) ic was already present in G, and hence $(z, end(ic)) \in \varphi(u)$, or (3) ic is a new information channel. Using similar arguments as in the proof of Lemma 1, we can show that ic needs to start with the new interaction and that the remainder of ic forms an information channel ic' from v to z in G with $end(ic') = end(ic)$. In that case $(z, end(ic)) \in \varphi(v)$. Given the constraint on duration we furthermore need to have $end(ic) - t + 1 \leq \omega$. Hence, $\varphi'(u)$ needs to be a subset of $\{(v,t)\} \cup \varphi(u) \cup \{(z,t') \in \varphi(v) \mid t' - t + 1 \leq \omega\}$, and we can obtain $\varphi'(u)$ by only keeping those pairs that are not dominated. □

Example 2. Figure 1a represents a small interaction network and Table 1b shows the edges in order of time. For $\omega = 3$ the Influence Summary Set will update as follows:

	a	b	c	d	e	f
φ	{}	{}	{}	{}	{}	{}

	a	b	c	d	e	f
$\xrightarrow{(b,c,8)}$ φ	{}	**(c, 8)**	{}	{}	{}	{}

	a	b	c	d	e	f
$\xrightarrow{(e,c,7)}$ φ	{}	(c, 8)	{}	{}	**(c, 7)**	{}

	a	b	c	d	e	f
$\xrightarrow{(b,e,6)}$ φ	{}	**(c, 7)(e, 6)**	{}	{}	(c, 7)	{}

	a	b	c	d	e	f
$\xrightarrow{(a,b,5)}$ φ	**(b, 5)** **(c, 7)** **(e, 6)**	(c,7) (e,6)	{}	{}	(c, 7)	{}

	a	b	c	d	e	f
$\xrightarrow{(e,b,4)}$ φ	(b,5) (c,7) (e,6)	(c,7) (e,6)	{}	{}	(c,7) **(b, 4)**	{}

	a	b	c	d	e	f
$\xrightarrow{(d,e,3)}$ φ	(b,5) (c,7) (e,6)	(c,7) (e,6)	{}	**(e, 3)** **(b, 4)**	(c,7) (b,4)	{}

	a	b	c	d	e	f
$\xrightarrow{(e,f,2)}$ φ	(b,5) (c,7) (e,6)	(c,7) (e,6)	{}	(e,3) (b,4)	(c,7) (b,4) **(f, 2)**	{}

	a	b	c	d	e	f
$\xrightarrow{(a,d,1)}$ φ	(b,5) (c,7) **(e, 3)** **(d, 1)**	(c,7) (e,6)	{}	(e,3) (b,4)	(c,7) (b,4) (f,2)	{}

While processing the edge $(b,e,6)$, first we add $(e,6)$ in the summary of d and then add $(c,7)$ from the summary of e in summary of b. As the summary of b already had $(c,8)$, the value will be updated. Next, during the processing of edge $(a,b,5)$ the summary of a is updated first by adding $(b,5)$ then while merging the summary of b in a we will ignore $(e,8)$ because the duration of the channel is 4 and the permitted window length is 3. The only addition is hence $(c,7)$.

THEOREM 1. *Algorithm 2 updates the IRS summary correctly.*

PROOF. This proof follows by induction. For the empty list of transactions, the algorithm produced the empty summary. This is our base case. Then, for every interaction that is added in the for loop, it follows from Lemma 1 and Lemma 2 that the summaries are correctly updated to form the summary of the interaction graph with one more (earlier) interaction. After all interactions have been processed, the summary is hence that of the complete interaction graph. □

Lemma 3. Algorithm 2 runs in time $\mathcal{O}(mn)$ and space $\mathcal{O}(n^2)$, where $n = |V|$ and $m = |\mathcal{E}|$.

PROOF. Each edge in $\mathcal{E}$ is processed exactly once and for each edge, both ADD and MERGE are called once. We assume that the summary sets $\varphi(u)$ are implemented with hash tables such that looking up the element (v,t) for a given v takes constant time only. Under this assumption, the ADD function has constant complexity. The MERGE function calls ADD for every item in $\varphi(v)$ at least once. The number of items in $\varphi(v)$ is upper bounded by n and hence the time complexity of one merge operation is at most $\mathcal{O}(n)$. This leads to the upper bound $\mathcal{O}(mn)$ in total.

For the space complexity, note that in the worst case for each node there is an information channel to every other node of duration at most ω. In that case, the size of the individual summary $\varphi(v)$ of every node v is $\mathcal{O}(n)$ which leads to a space complexity of $\mathcal{O}(n^2)$ in total. □

As we can see from Lemma 3 the memory requirements for the exact algorithm is in worst case quadratic in the

Algorithm 2 Influence set with Exact algorithm

```
Input: Interaction graph $G(V,\mathcal{E})$. $\ell_G$ is the list of interactions reversely ordered by time stamp
    Threshold $\omega$ (maximum allowed duration of an influence channel)
Output: $\varphi(u)$ for all $u \in V$

  function ADD($\varphi(u)$,$(v,t)$)
      if $\exists t^{'} : (v,t^{'}) \in \varphi(u)$ then
                                   ▷ There is at most one such entry
          if $t < t^{'}$ then
              $\varphi(u) = (\varphi(u) \setminus (v,t^{'})) \cup (v,t)$
          end if
      else
          $\varphi(u) = \varphi(u) \cup \{(v,t)\}$
      end if
  end function

  function MERGE($\varphi(u)$,$\varphi(v)$,$t$,$\omega$)
      for all $(x,t_x) \in \varphi(v)$ do
          if $t_x - t < \omega$ then ADD($\varphi(u)$,$(x,t_x)$)
          end if
      end for
  end function

  Initialize: $\varphi(u) \leftarrow \emptyset \quad \forall u \in V$
  for all $(u,v,t) \in \ell_G$ do
      ADD($\varphi(u)$,$(v,t)$)
      MERGE($\varphi(u)$,$\varphi(v)$,$t$,$\omega$)
  end for
```

number of nodes of the graph. This will not scale well for large graphs as we want to keep this data structure in memory for efficient querying. Hence in the next section we will present an approximate but more memory and time efficient version of the algorithm.

3.2 Approximate Algorithm

Algorithm presented in the previous section computes the *IRS* exactly, albeit at the cost of high space complexity and update time. In this section, we describe an approximate algorithm which is much more efficient in terms of memory requirements and update time. The approximate algorithm is based on an adaptation of the HyperLogLog sketch [9].

3.2.1 HyperLogLog Sketch

A HyperLogLog (HLL) sketch [9] is a probabilistic data structure for approximately counting the number of distinct items in a stream. Any exact solution for counting the number of distinct items in a stream would require $\mathcal{O}(N)$ space with N the cardinality of the set. The HLL sketch, however, approximates this cardinality with no more than $\mathcal{O}(\log(\log(N)))$ bits. The HLL sketch is an array with $\beta = 2^k$ cells $(c_1, \ldots, c_\beta)$, where k is a constant that controls the accuracy of the approximation. Initially all cells are 0. Every time an item x in the stream arrives, the HLL sketch is updated as follows: the item x is hashed deterministically to a positive number $h(x)$. The first k bits of this number determines the 0-based index of the cell in the HLL sketch that will be updated. We denote this number $\iota(x)$. For the remaining bits in $h(x)$, the position of the least significant bit that is 1 is computed. This number is denoted $\rho(x)$. If $\rho(x)$ is larger than $c_{\iota(x)}$, $c_{\iota(x)}$ will be overwritten with $\rho(x)$.

For example, suppose that we use a HLL sketch with $\beta = 2^2 = 4$ cells. Initially the sketch is empty:

0	0	0	0

Suppose now item a arrives with $h(a) = 1110100110010110_b$. The first 2 bits are used to determine $\iota(a) = 11_\beta = 3$. The rightmost 1 in the binary representation of $h(a)$ is in position 2, and hence c_3 becomes 2. Suppose that next items arrive in the stream with $(c_{\iota(x)}, \rho(x))$ equal to: $(c_1, 3)$, $(c_0, 7)$, $(c_2, 2)$, and $(c_1, 2)$, then the content of the sketch becomes:

7	3	2	2

It is clear that duplicate items will not change the summary. Furthermore, for a random element x, $P(\rho(x) \geq \ell) = 2^{-\ell}$. Hence, if d different items have been hashed into cell c_ι, then $P(c_\iota \geq \ell) = 1 - (1 - 2^{-\ell})^d$. This probability depends on d, and all c_i are independent. Based on a clever exploitation of these observations, Flajolet et al. [9] showed how the number of distinct items in a stream can be approximated from the HLL sketch. Last but not least, two HLL sketches can easily be combined into a single sketch by taking for each index the maximum of the values in that index of both sketches.

3.2.2 Versioned HLL Sketch

The HLL sketch is an excellent tool for our purpose; every time an edge (a, b) needs to be processed (recall that we process the edges in reverse chronological order), all nodes reachable by an information channel from b, are also reachable by an information channel from a. Therefore, if we keep the list of reachable nodes as a HLL sketch, we can update the reachable nodes from a by unioning in the HLL sketch of the reachable nodes from b into the HLL sketch of those reachable from a. One aspect, however, that is not taken into account here is that we only consider information channels of length ω. Hence, only those nodes reachable from b by an information channel that ends within time window ω should be considered. Therefore, we developed a so-called *versioned* HLL sketch vHLL. The vHLL maintains for each cell c_i of the HLL a list L_i of $\rho(x)$-values together with a timestamp and is updated as follows: let $t_{current}$ be the current time; periodically entries (r, t) with $t - t_{current} + 1 > \omega$ are removed from vHLL. Whenever an item x arrives, $\rho(x)$ and $\iota(x)$ are computed, and the pair $(\rho(x), t_{current})$ is added to the list $L_{\iota(x)}$. Furthermore, all pairs (r, t) such that $r \leq \rho(x)$ are removed from $L_{\iota(x)}$. The rationale behind the update procedure is as follows: at any point in time $t_{current}$ we need to be able to estimate the number of elements x that arrived within the time interval $[t_{current}, t_{current} + \omega - 1]$. Therefore it is essential to know the maximal $\rho(x)$ of all x that arrived within this interval. We keep those pairs (r, t) in L_ι such that r may, at some point, become the maximal value as we shift the window further back in time. It is easy to see that any pair (r, t) such that $r \leq \rho(x)$ for a newly arrived x at $t_{current}$ will always be dominated by $(\rho(x), t_{current})$. On the other hand, if $\rho(x) < r$ we still do have to store $(\rho(x), t_{current})$ as (r, t) will leave the window before $(\rho(x), t_{current})$ will.

Example 3. Suppose that the elements e, d, c, a, b, a have to be added to the vHLL. Recall that we process the stream in reverse order, hence the updates are processed in the following order: (a, t_6), (b, t_5), (a, t_4), (c, t_3), (d, t_2),

(e, t_1). Let ι and ρ be as follows for the elements in V:

item	a	b	c	d	e
ι	1	3	3	2	2
ρ	3	1	2	2	1

The subsequent vHLL sketches are respectively the following:

	$\{\}$	$\{\}$	$\{\}$	$\{\}$
$\xrightarrow{(a,t_6)}$	$\{\}$	$(3,t_6)$	$\{\}$	$\{\}$
$\xrightarrow{(b,t_5)}$	$\{\}$	$(3,t_6)$	$\{\}$	$(1,t_5)$
$\xrightarrow{(a,t_4)}$	$\{\}$	$(3,t_4)$	$\{\}$	$(1,t_5)$
$\xrightarrow{(c,t_3)}$	$\{\}$	$(3,t_4)$	$\{\}$	$(2,t_3)$
$\xrightarrow{(d,t_2)}$	$\{\}$	$(3,t_4)$	$(2,t_2)$	$(2,t_3)$
$\xrightarrow{(e,t_1)}$	$\{\}$	$(3,t_4)$	$(2,t_2),(1,t_1)$	$(2,t_3)$

Notice that also two vHLL sketches can be easily combined by merging them. For each cell ι, we take the union of the respective lists L_ι and L'_ι and remove all pairs (r,t) in the result that are dominated by a pair (r',t') that came from the other list with $t' < t$ and $r' \geq r$. If the lists are stored in order of time, this merge operation can be executed in time linear in the length of the lists.

Example 4. Consider the following two vHLL sketches:

$\{\}$	$(3,t_4)$	$(1,t_1),(2,t_2)$	$(2,t_3)$

$\{(5,t_1)\}$	$(3,t_2)$	$(4,t_3)$	$(1,t_4)$

The result of merging them is:

$\{(5,t_1)\}$	$(3,t_2)$	$(1,t_1),(2,t_2),(4,t_3)$	$(2,t_3)$

Note that adding versioning to the HLL sketch comes at a price.

Lemma 4. The expected space for storing a vHLL sketch for a window length ω is $\mathcal{O}(\beta(\log(\omega)^2))$.

PROOF. The size of each pair (r,t) stored in a list L_ι is dominated by t and takes space $\mathcal{O}(\log(\omega))$. In worst case, all elements in the window $x_{current}, \ldots, x_{current+\omega-1}$ are different and all arrive into the same cell c_ι. In that case, the expected number of pairs in $L\iota$ is $E[X_1 + X_2 + \ldots + X_{\omega-1}]$ where X_i denotes the following statistical variable: X_i equals 1 if $(\rho(x_i), t_{current+i-1})$ is in $L\iota$ and 0 otherwise. This means that $X_i = 1$ if and only if $\rho(x_i) > \max\{\rho(x_1), \ldots, \rho(x_{i-1})\}$. As each $\rho(x_j)$, $j \leq i$ has the same chance to be the largest, $P(X_i = 1) \leq \frac{1}{i}$. Hence we get:

$$E[|L_\iota|] \leq E[X_1 + \ldots + X_{\omega-1}] \leq \sum_{i=1}^{\omega} \frac{1}{i} = \mathcal{O}(\log(\omega)) \ .$$

□

3.2.3 *vHLL-Based Algorithm*

The approximate algorithm is very similar to the exact algorithm 2; instead of using exact sets we use the more compact versioned HyperLogLog sketch. ADD and MERGE are the only functions which need to be updated as per the new sketch everything else will remain the same as shown in algorithm 2. We will just present the APPROXADD and APPROXMERGE functions in Algorithm 3.

Lemma 5. The expected time complexity for Algorithm 3 is $\mathcal{O}(m\beta(\log(\omega))^2)$, where $n = |V|$ and $m = |\mathcal{E}|$.

Algorithm 3 Approximate Algorithm for IRS

```
function APPROXADD($\varphi(u)$,($\rho(v), t$),$\iota(v)$)
    if $\exists(\rho, t') \in L_\iota : (\rho, t')$ dominates $(\rho(v), t)$ then
        Ignore $(\rho(v), t)$
    else
        if $\exists(\rho, t') \in L_\iota : (\rho(v), t)$ dominates $(\rho, t')$ then
            remove $(\rho, t')$ from $L_\iota$
        end if
        Append $(\rho(v), t)$ in $L_\iota$
    end if
end function
function APPROXMERGE($\varphi(u)$,$\varphi(v)$,$t$,$\omega$)
    while $i < \beta$ do
        for all $(x, t_x) \in L_i$ do            ▷ Iterate over $\varphi(v)$
            if $t_x - t < \omega$ then
                APPROXADD($\varphi(u)$,$(x, t_x)$, $i$)
            end if
        end for
        $i++$
    end while
end function
```

PROOF. In the APPROXMERGE function the while loop will run for β iterations and the inner for loop will run for an expected of $\log(\omega)$ items(from Lemma 4). Hence time complexity would be $\mathcal{O}(\beta\log(\omega)\mathcal{O}(\text{APPROXADD}))$.
Now in the APPROXADD function there are at-most $\log(\omega)$ comparisons, hence $\mathcal{O}(\text{APPROXADD}) = \mathcal{O}(\log(\omega))$. For each edge APPROXADD and APPROXMERGE are called only once. Hence $\mathcal{O}(m\beta(\log(\omega))^2)$ is the expected time complexity. □

Lemma 6. The expected space complexity for the Algorithm 3 is $\mathcal{O}(n\beta(\log(\omega))^2)$, where $n = |V|$ and $m = |\mathcal{E}|$.

PROOF. From Lemma 4 the expected size of one vHLL sketch is $\mathcal{O}(\beta(\log(\omega))^2)$. There will be only one vHLL sketch for each node, hence, expected space complexity is $\mathcal{O}(n\beta(\log(\omega))^2)$. □

4. APPLICATIONS

4.1 Influence Oracle:

Given the *Influence Reachability Set* of an interaction network computing the influence spread of a given seed set, $S \subseteq V$ is straightforward. The influence spread for seed set S is computed as:

$$Inf(S) = \bigcup_{u \in S} \sigma(u) \qquad (1)$$

HyperLogLog sketch union requires taking the maximum at each bucket index ι which is very efficient, so the the time complexity would be $\mathcal{O}(|S|\ell)$.

4.2 Influence Maximization:

Influence Maximization deals with the problem of finding top k seed nodes which will maximize the influence spread. After the pre processing stage of computing *IRS* we can use a greedy approach to find the top-k seed nodes by using the Influence oracle. First we show the complexity of the top-k most influential nodes problem is NP-hard and then show that the Influence oracle function is monotone and submodular. Hence we can use a greedy approximation approach.

Lemma 7. Influence maximization under the *Influence Reachability Set* model is NP-hard.

PROOF. Given the *Influence Reachability Set* for all the nodes the problem of finding a subset of k nodes such that the union is maximum is a problem which is similar to the problem of maximum coverage problem. As the later is a NP-hard problem we deduce that the given problem is NP-hard. $\square$

Lemma 8. The influence function $\sigma(S)$ is submodular and monotone.

PROOF. First we will prove that $Inf(S)$ is a submodular function. Let S and T be two sets of seed nodes such that $S \subset T$. Let x be another node not in T. Now, let the marginal gain of adding x in S, i.e., $Inf(S + x) - Inf(S) = P$. P is the set of those nodes for which there is no path from S and hence these should belong to $Inf(x)$. Let the marginal gain of adding x in T,i.e., $Inf(T+x) - Inf(T) = P'$. It is clear that $P' \subseteq P$, as otherwise there will be a node u for which there is a path from S but not from T and this is not possible given $S \subset T$. Hence $Inf(S + x) - Inf(S) \geq Inf(T + x) - Inf(T)$.

It is obvious to see the that *Inf* is monotone as it is a increasing function, adding a new node in the seed set will never decrease the influence, and hence if $S \subset T$ then $Inf(S) \leq Inf(T)$ $\square$

Greedy Approach for Influence Maximization:

Algorithm 4 outlines the details for the greedy approach. We start by first sorting the nodes based on the size of the *Influence Reachability Set*. The node with maximum IRS set size becomes the most influential node and is taken as the first node in seed set. Next at each stage we iterate through the sorted list and check the gain by using influence oracle of the already selected nodes and the new node. The node which results in maximum gain is added into the seed set.

Algorithm 4 Influence Maximization using IRS

```
Input: The Influence set $\sigma_u \forall u \in V$ and the number of
  seed nodes to find is k
  initialize $selected \leftarrow \emptyset \wedge covered \leftarrow \emptyset$
  Sort $u \in V$ descending with respect to $|\sigma_u|$. Save this
  sorted list as $\ell$
  while $selected < k$ do
      $gain = 0$ ; $u_s = \emptyset$
      for all $u \in \ell$ do
          if $|covered \cup \sigma_u| - |covered| > gain$ then
              $gain = |covered \cup \sigma_u| - |covered|$
              $u_s = \{u\}$
          end if
          if $gain > \sigma_u$ then
              break;
          end if
      end for
      $selected \leftarrow selected \cup u_s$; $covered \leftarrow covered \cup \sigma_{u_s}$
  end while
```

5. RELATED WORK

The problem of Influence Maximization and Influence spread prediction is a well know problem. Broadly, the work in this area can be categorized into two main categories. The first category is based on static graphs [7, 23, 13, 6] where the underlying graph is already given and the probability of a node getting influenced is derived from probabilistic simulations. The second category is data driven, where the underlying influence graph is derived based on a relationship such as friendship between two users or common action within a specified time [24, 8, 11, 10]. The static graph approaches do not capture the dynamics of real networks such as social media and hence the data driven approaches are more suitable.

Static graph.

The Influence Maximization problem in social network was first studied by Richardson et al. [7, 23] where they formalized the problem with a probabilistic model. Later Kempe et al. [13] proposed a solution using discrete optimization. They proved that the Influence Maximization problem is NP-hard and provided a greedy algorithm to select seed sets using maximum marginal gain. As the model is based on Monte Carlo simulations, it is not scalable for large graphs. Later improvements were proposed by Chen et al. [4] using the DegreeDiscountand *prefix excluding maximum influence in-arborescence* (PMIA) [3] algorithms. Both algorithms are heuristic-based. Leskovec et al. proposed the Cost-Effective Lazy Forward (CELF) [17] mechanism to reduce the number of simulations required to select seeds. All of the above-mentioned studies focus on static graph and do not take the temporal nature of the interactions between different nodes into consideration. The latest work on the static graph Influence Maximization problem by Cohen et al. [6] is the fastest we have come across which scales to very large graphs. We compare our seed sets and their influence spread with the seeds selected by their algorithm SKIM. Related work on information flow mining on static graph may be found in [14, 17, 19, 22, 21]. Lie et al. in [20] and Chen et al. in [2] independently proposed the first time constrained Influence Maximization solutions for static graph. Their work considers the concept of time delay in information flow. They assign this delay at individual node level based on different probabilistic models and not the information channels or pathways between the nodes.

Data Driven approach.

There are a few recent work which consider the temporal aspect of the graph and are based on real interaction data. Goyal et al. [11] proposed the first data based approach to find influential users in a social network by considering the temporal aspect in the cascade of common actions performed by users, instead of using just static simulation of the friendship network. However, their work does not consider the time constraint in the information flow. In [10] they do use a time window based approach to determine true leaders in the network. However, the time window they consider is for direct influence only, i.e., once a user performs an action how many of his/her friends repeat that action in that time window. They have some additional assumptions like information propagation is non-cyclic and if one user performs an action more then once, they use only the time stamp of the first action. Our approach does not make such assumptions and identifies influential nodes without any constraints on the number of times a user performs an action or that the propagation graph needs to be a DAG. The time constraints we impose are on the path of information flow from the start of the action. Also, our proposed solu-

Table 1: Comparison of related work on different parameters

	Gomez-Rodriguez [24]	Cohen [6]	Du,N [8]	Tang [25]	Goyal [11, 10]	Kempe [13]	Lei [20]	IRS
Static Graph(S), Data or Cascade (C), Interaction Network (I)	C	S	C	S	C	S	S	I
Considers information channel or pathways?	Yes	No	Yes	No	Yes	No	No	Yes
Time window constrained	Yes	No	Yes	No	Yes	No	No	Yes
Approx sketching or sampling	Yes	Yes	Yes	Yes	No	No	Yes	Yes
One Pass algorithm	No	Yes	No	No	Yes	No	Yes	Yes

tion just needs a single pass over the propagation graph whereas Goyal's work do a single pass over the action log but multiple passes on the social network to find the child nodes. Our sketch based approximation further improves the time and space complexity.

There are a few more recent works on data driven approach by Gomez-Rodriguez et al. [24] and Du et al. [8]. These works try to derive the underlying hidden influence network and the influence diffusion probabilities along every edge from a given cascade of infection times for each node in the network. Du et al. [8] proposed a scalable algorithm called *ConTinEst*, which finds most influential nodes from the derived influence network. *ConTinEst* uses an adaption of a randomized neighborhood estimation algorithm [5] to find the most influential node in the network. But getting the cascade data of infection times for every network is not always possible. For example in an email or a messaging network, we may have access only to interactions between the users and not to the actual individual infection time. To the best of our knowledge our work is the first to try to predict and maximize influence in a network in which only the interaction data is available and no other action cascade or relationship between users is provided.

In Table 1 we give a brief comparison matrix of our IRS approach with some of the other works in Influence Maximization. We compare against the type of input each approach considers; i.e, a static graph (S), action cascade or infection time based event cascades (C) or interaction network based (I). We also compare if in the modeling of the information propagation in the approach considers information pathways or channels to do influence maximization and if the pathways have time window based constrains. For performance comparison, we see if they do use some sampling or sketching techniques to improve performance and if the algorithm is a one pass algorithm.

6. EXPERIMENTAL EVALUATION

In this section, we address the following questions:

Accuracy of Approximation. How accurate is the approximation algorithm for the Oracle problem? In other words, how well can we estimate the size of the IRS set based on the versionned HLL sketch?

Efficiency. How efficient is the approximate algorithm in terms of processing time per activity, and how does the window length ω impact the efficiency? How long does it take to evaluate an Oracle query based on the IRS summary?

Effectiveness. How effective is the identification of influential nodes using IRS to maximize the influence spread under the Time-Constrained Information Cascade Model? To this end, we compare our algorithm to a number of competitors:

- SKIM is the only algorithm which scale to large datasets in few minutes time. We ran SKIM using the same parameters Cohen et al. [6] use in their paper for all the experiments. SKIM is from the category of algorithms which considers a static graph and takes input in the form of a DIAMICS format graph. Hence we convert the interaction network data into the required static graph format by removing repeated interactions and the time stamp of every interaction.
- ConTinEst(CTE) [8] is the latest data driven algorithm which works on static networks where the edge weights corresponds to the associated transmission times. The edge weight is obtained from a transmission function which in turn is derived from an cascade of infection time of every node. As we assume that only the interaction between different nodes of a network is being observed and no other information such as the Infection time cascade is available, we transform the interactions into a static network with edge weights as required by ConTinEst. The first time a node u appears as the source of an interaction we assign the infection time u_i for the source node as the interaction time. Then each interaction (u, v, t) is transformed into an weighted edge (u, v) with the edge weight as the difference of the interaction time and the time when the source gets infected, i.e, $t - u_i$. We ran the same code as published by the authors with the default settings on the transformed data.
- The popular baselines *PageRank(PR)* and *High Degree(HD)*[13]. Here we select the k nodes with respectively the highest PageRank and out-degree. Notice that for PageRank we reversed the direction of the interaction edges, as PageRank measures incoming "importance" whereas we need outgoing "influence." By reversing the edges this aspect is captured. To make a fair comparison with our algorithm that takes into account the overlap of the influence of the selected top-influencers, we developed a version of HD that takes into account overlap. That is, we select a set of nodes that *together* have maximal outdegree. In our experiments we call this method the Smart High Degree approach (SHD). Notice that SHD is actually a special case of our IRS algorithm, where we set $\omega = 0$.

We also ran some performance experiments comparing the competitors to our IRS algorithm. In the interpretation of these results, however, we need to take into ac-

Table 2: Characteristics of interaction network along with the time span of the interactions as number of days.

Dataset	$\|\mathcal{V}\|[.10^3]$	$\|\mathcal{E}\|[.10^3]$	Days
`Enron`	87.3	1,148.1	8,767
`Lkml`	27.4	1,048.6	2,923
`Facebook`	46.9	877.0	1,592
`Higgs`	304.7	526.2	7
`Slashdot`	51.1	140.8	978
`US-2016`	4,468	44,638	16

count that the static methods require the graph to be pre-processed and takes as input the flattened non-temporal graph, which is in some cases significantly smaller as it does not take repetitions of activities into account.

6.1 Datasets and Setup

We ran our experiments on real-world datasets obtained from the SNAP repository [18] and the koblenx network collection [16]. We tested with *social* (`Slashdot`, `Higgs`, `Facebook`) and *email* (`Enron`, `Lkml`) networks. As the real world interaction networks available from previous works were not large enough to test the scalability of our algorithm, we created another dataset by tracking tweets related to the US Election 2016. We follow the same technique used to create the `Higgs` data set of the SNAP repository. Statistics of these data sets are reported in Table 2. These datasets are available online, sorted by time of interaction. We kept the datasets in this order, as our algorithm assumes that the interactions are ordered by time. This assumption is reasonable in real scenarios because the interactions will always arrive in increasing order of time and it is hence plausible that they are stored as such. The overall time span of the interactions varies from few days to many years in the data sets. Therefore, in our experiments we express the window length as a percentage of the total time span of the interaction network.

The performance results presented in this section are for the C++ implementation of our algorithm. All experiments were run on a simple desktop machine with an Intel Core i5-4590 CPU @3.33GHz CPU and 16 GB of RAM, running the Windows 10 operating system. For the larger dataset `US-2016` the memory required was more than 16 GB. hence, we ran the experiments for the `US-2016` dataset on a Linux system with 64 GB of RAM.

6.2 Accuracy of the Approximation

In order to test the accuracy of the approximate algorithm, we compared the algorithm with the exact version. We compute the average relative error in the estimation of the *IRS* size for all the nodes, in function of the number of buckets ($\beta = 2^k$). Running the exact algorithm is infeasible for the large datasets due to the memory requirements, and hence, we test only on the `Slashdot` and `Higgs` datasets to measure accuracy. We tested accuracy at different window lengths. The results are reported in Table 3. As expected from previous studies, the accuracy increases with β. There is a decrease in accuracy with increasing window length because as the window length increases, the number of nodes with larger *IRS* increases as well, resulting in a higher average error. β values beyond 512 yield only modest further improvement in the accuracy. Therefore, we used $\beta = 512$ as default for all of the next experiments.

Table 3: Average relative error in the estimation of the *IRS* size for all the nodes as a function of b for different window length.

Dataset	β	window %		
		1	10	20
`Higgs`	16	0.075	0.116	0.113
	32	0.044	0.081	0.053
	64	0.026	0.056	0.046
	128	0.008	0.015	0.017
	256	0.005	0.008	0.009
	512	0.002	0.006	0.007
`Slashdot`	16	0.048	0.055	0.105
	32	0.023	0.044	0.042
	64	0.013	0.022	0.33
	128	0.011	0.04	0.05
	256	0.01	0.026	0.025
	512	0.005	0.019	0.02

Table 4: Memory used in MB to process all the interactions at different window length ω

Datasets	$\omega = 1$	$\omega = 10$	$\omega = 20$
`Slashdot`	194.9	385.4	431.5
`Higgs`	1008.6	1138.3	1229.8
`Enron`	416.3	426	426.3
`Facebook`	247.4	470	496.2
`Lkml`	228.5	282.5	295.2
`US-2016`	50,449	56,829	59,104

6.3 Runtime and Memory usage of the Approximation Algorithm

We study the runtime of the approximation algorithm on all the datasets for different window lengths ω. The runtime increases with the increasing window length, as expected given that the number of nodes in the *IRS* increases, resulting in more elements in the vHLL to be merged. We study the processing time in function of the time window ω. Here we vary ω from 1% to 100%. The results are reported in Figure 3. It is interesting to see in Figure 3 that the processing time becomes almost constant as soon as the window length reaches 10%. This is because the *IRS* does not change much once the time window is large enough. This behavior indicates that at higher window lengths the analysis of the interaction network becomes similar to that of the underlying static network. As the algorithm is one pass it scales linearly with the input size. For the largest data set `US-2016` with approx 45 million interactions the algorithm was able to parse all the interactions in just 8 min.

As shown in Table 4, we observe that the space consumption is essentially dependent on the number of nodes and not on the number of interactions on the network. For example, on `Enron` dataset the total space requirement is just 295 MB for $\omega = 20\%$, whereas for `Higgs` the memory requirement is 1229 MB, as the number of nodes for this data set is 4 times that of `Enron`. It is natural to see a slight increase in the space requirement with window length ω as the lists in the vHLL sketches become larger.

6.4 Influence Oracle Query Efficiency

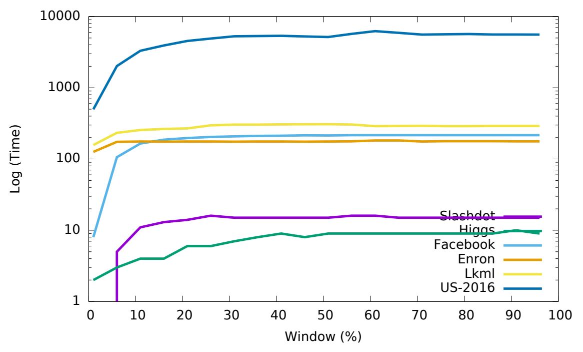


Figure 3: Log of the time to process all the interactions as a function of time window ω

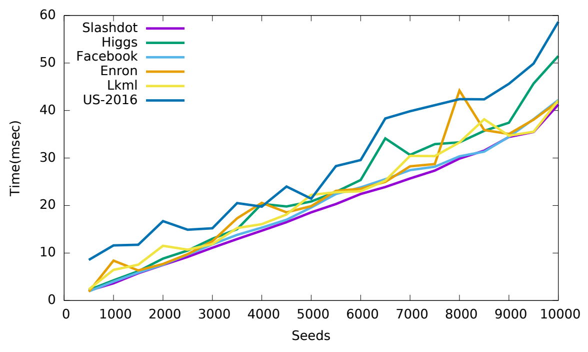


Figure 4: Influence spread prediction query time in milliseconds for window length, $\omega = 20\%$ as a function of the seed set size.

Now, we present the query time for the Influence Oracle using *IRS*. After the pre-processing step of computing the *IRS* for all nodes, querying the data structure is very efficient. We pick seed nodes randomly and query the data structure to calculate their combined influence spread. In Figure 4 we report the average query time for randomly selected seeds. We observe that, irrespective of the graph size the query time is mostly the same for all graphs. This is because the complexity of the versionned HyperLogLog union is independent of the set size. As expected, query time increases with the number of seed nodes. Even for numbers of seed nodes as large as 10, 000, the query time is just few milliseconds.

6.5 Influence Maximization

Our next goal is to study how the *Influence Reachability Set* could be used to solve the problem of Influence Maximization. First we do an effectiveness analysis and then an efficiency comparison with the baseline approaches.

Effectiveness analysis:

We compare the influence spread by running the Time Constrained Information Cascade Model with infection probabilities of 50% and 100%. We compare our sketch based algorithm with the latest sketch based probabilistic approach SKIM [6] and ConTinEst(CTE) [8]. As Both SKIM and ConTinEst require a specific input format of the underlying static graph we ran a pre-processing phase to generate the required graph data from the interaction network. We ran both SKIM and ConTinEst using the code published by the respective authors. We also

Table 5: Common seeds between different window length for top 10 seeds

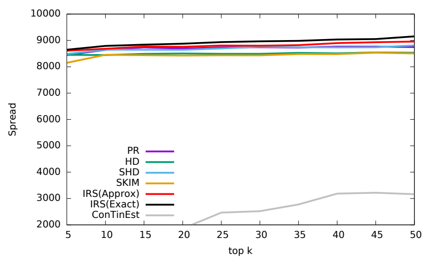
Datasets	1% - 10%	1% - 20%	10% - 20%
Slashdot	0	0	7
Higgs	3	1	3
Enron	0	0	6
Facebook	4	4	9
Lkml	1	0	5
US-2016	6	6	10

compare with other popular baselines *PageRank(PR)* and *High Degree(HD)*[13] by selecting top k nodes with highest page rank and highest out degree. We used 0.15 as the restart probability and a difference of 10^{-4} in the $L1$ norm between two successive iterations as the stopping criterion. We also introduced a variation of High Degree called *Smart High Degree(SHD)* in which instead of selecting top k nodes with highest degree we select nodes using a greedy approach to maximize the distinct neighbors.

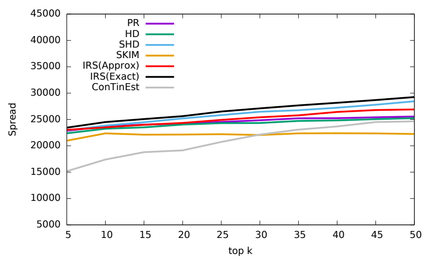
The results of our comparison are reported in Figure 5. We observe that in all the datasets the influence spread by simulation through the seed nodes selected by our IRS exact algorithm is consistently better than that of other baselines. The IRS approx approach results in lesser spread but still it is best for Lkml dataset and is close to other baselines in other datasets. In other datasets like Enron or Facebook the nodes with highest degree are the same node for which the longer temporal paths exists hence the spread is similar. SKIM and ConTinEst both perform worst at smaller windows but with higher window lengths their performance increases; this is because for higher window lengths there is less pruning of the information channels resulting in a very small change in the Influence reachability set size. Hence, the behavior is the same as the analysis of the static graph and the time window does not have much effect on the *Influence Reachability Set*. The Smart High Degree approach out-performs High Degree in all of the cases. For smaller values of k the spread is very similar because of common seeds, for example 4 out of 5 seeds are common in Slashdot as nodes with highest page Rank is the also the node with highest degree and highest IRS set size at $\omega = 1\%$. But as k increases IRS performs much better.

Efficiency analysis:

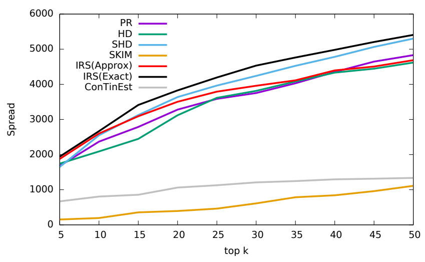
Next, we compared the time required to find the top 50 seeds. The results are reported in Table 6. For IRS we report time taken by the more efficient IRS approx approach. The IRS approach takes more time for Enron and Lkml as compare to other baselines because the IRS approach depends on the number of interactions. While IRS is slower than Page Rank and Smart High Degree for smaller datasets it scales linearly with the size and takes 8 times less time for the US-2016 dataset with millions of nodes and interactions. For SKIM the time required to find top k seeds is quite low. However, it requires preprocessed data in the DIMACS graph format [1] and the pre-processing step takes up to 10 hours for the US-2016 dataset. ConTinEst does not scale so well for large graphs and is the slowest in all dataseta. For the US-2016 dataset the memory requirements were so high that it could not even finish the processing. IRS provides a promising tradeoff between efficiency and effectiveness, especially for smaller window lengths when the tempo-



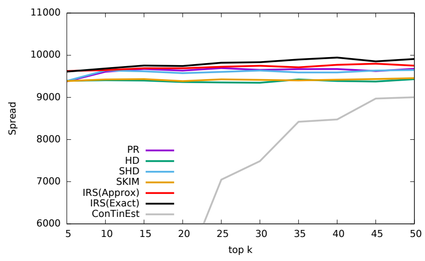
(a) Lkml ($\omega = 1\%$)



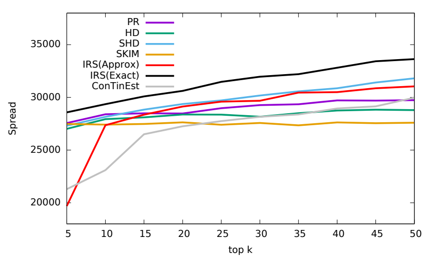
(b) Enron ($\omega = 1\%$)



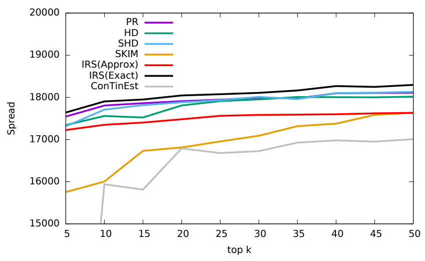
(c) Facebook ($\omega = 1\%$)



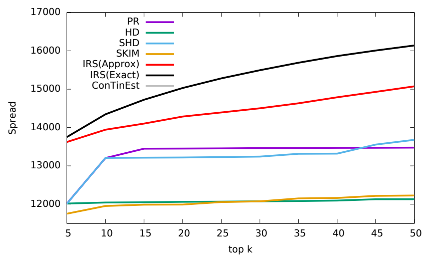
(d) Lkml ($\omega = 20\%$)



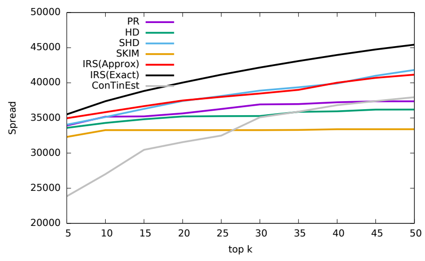
(e) Enron ($\omega = 20\%$)



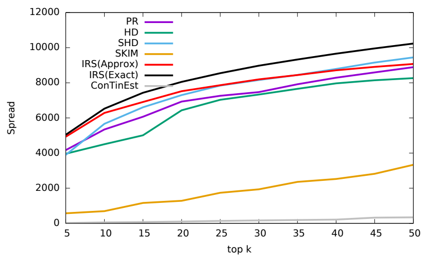
(f) Facebook ($\omega = 20\%$)



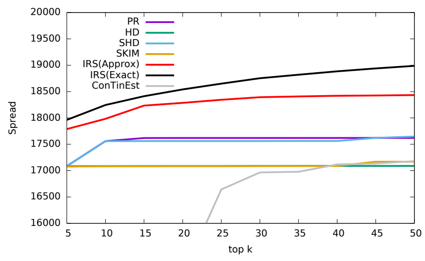
(g) Lkml ($\omega = 1\%$)



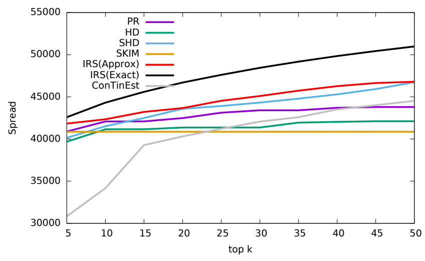
(h) Enron ($\omega = 1\%$)



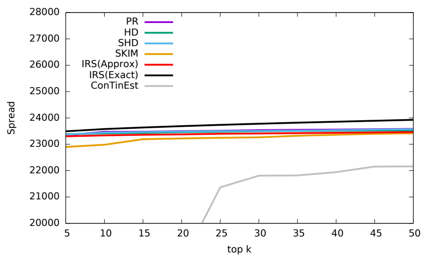
(i) Facebook ($\omega = 1\%$)



(j) Lkml ($\omega = 20\%$)



(k) Enron ($\omega = 20\%$)



(l) Facebook ($\omega = 20\%$)

Figure 5: Comparing the spread of the influence of top k seeds using Simulation Algorithm for different seed size at different window length ω at Infection probability 50%(a-f) and 100%(g-l) respectively.

Table 6: Time in seconds to find top 50 seeds by IRS(approx) and all other baseline approach.

Datasets	IRS	SKIM	PR	HD	SHD	CTE
Slashdot	1.1	1.2	21.9	0.9	2.1	694
Higgs	2.2	4.3	29.8	0.7	1.5	3,802
Enron	93.7	2.2	49.4	0.4	8.1	1,349
Facebook	10.3	1.1	35.6	0.5	2.9	790
Lkml	117.9	1.7	29.8	0.5	22.9	733
US-2016	498	23.6	4,261	47.4	3,338.4	-

ral nature of the graph has a higher role in determining the influential nodes.

Effect of window on top k seeds:

To see the effect of the time window on the most influential nodes we study the common seeds between different window lengths. We observed that the top k seeds change drastically as we change the window length, especially when the window length is small. But for window lengths greater than 10% the top k seeds do not change much. For US-2016 the top 10 seeds are exactly the same for the 10% and 20% window. In Table 5 we have reported the common seeds among different top 10 seeds at different window lengths. There are no common seeds between the top 10 seeds found for window lengths of 1% and 10% for Slashdot and Enron and only $3-4$ common seeds for Higgs, Facebook and Lkml. This shows that for different window lengths there are different nodes which become most influential and hence it is necessary to consider window length while doing Influence maximization.

7. CONCLUSION

We studied the problem of information propagation in an interaction network—a graph with a sequence of time stamped interactions. We presented a new time constrained *influence channel* based approach for Influence Maximization and Information Spread Prediction. We presented an exact algorithm, which is memory inefficient, but it set the stage for our main technique, an approximate algorithm based on a modified version of HyperLogLog sketches, which requires logarithmic memory per network node, and has fast update time. One interesting property of our sketch is that the query time of the Influence Oracle is almost independent of the network size. We showed that the time taken to do influence maximization by a greedy approach on our sketch is very time efficient. We also showed the effect of the time window on the influence spread. We conclude that smaller window lengths have very high impact on the Information propagation and hence it is important to consider the spread window to do Influence maximization.

8. REFERENCES

[1] 9th DIMACS Implementation Challenge - Shortest Paths. http://www.dis.uniroma1.it/challenge9/format.shtml#graph, [Online; accessed 12-Sep-2016]

[2] Chen, W., Lu, W., Zhang, N.: Time-critical influence maximization in social networks with time-delayed diffusion process. arXiv:1204.3074 (2012)

[3] Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proceedings of the 16th ACM SIGKDD. pp. 1029–1038. ACM (2010)

[4] Chen, W., Wang, Y., Yang, S.: Efficient influence maximization in social networks. In: Proceedings of the 15th ACM SIGKDD. pp. 199–208. ACM (2009)

[5] Cohen, E.: Size-estimation framework with applications to transitive closure and reachability. Journal of Computer and System Sciences 55(3), 441–453 (1997)

[6] Cohen, E., Delling, D., Pajor, T., Werneck, R.F.: Sketch-based influence maximization and computation: Scaling up with guarantees. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. pp. 629–638. ACM (2014)

[7] Domingos, P., Richardson, M.: Mining the network value of customers. In: Proceedings of the seventh ACM SIGKDD. pp. 57–66. ACM (2001)

[8] Du, N., Song, L., Gomez-Rodriguez, M., Zha, H.: Scalable influence estimation in continuous-time diffusion networks. In: Advances in neural information processing systems. pp. 3147–3155 (2013)

[9] Flajolet, P., Fusy, É., Gandouet, O., Meunier, F.: Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. DMTCS Proceedings (2008)

[10] Goyal, A., Bonchi, F., Lakshmanan, L.V.: Discovering leaders from community actions. In: Proceedings of the 17th ACM conference on Information and knowledge management. pp. 499–508. ACM (2008)

[11] Goyal, A., Bonchi, F., Lakshmanan, L.V.: A data-based approach to social influence maximization. Proceedings of the VLDB Endowment 5(1), 73–84 (2012)

[12] Kempe, D., Kleinberg, J., Kumar, A.: Connectivity and inference problems for temporal networks. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing. pp. 504–513. ACM (2000)

[13] Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD. pp. 137–146. ACM (2003)

[14] Kleinberg, J.: The flow of on-line information in global networks. In: Proceedings of the 2010 ACM SIGMOD. pp. 1–2. ACM (2010)

[15] Kumar, R., Calders, T., Gionis, A., Tatti, N.: Maintaining sliding-window neighborhood profiles in interaction networks. In: Machine Learning and Knowledge Discovery in Databases, pp. 719–735. Springer (2015)

[16] Kunegis, J.: Konect: the koblenz network collection. In: Proceedings of the 22nd international conference on World Wide Web companion. pp. 1343–1350. International World Wide Web Conferences Steering Committee (2013)

[17] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost-effective outbreak detection in networks. In: Proceedings of the 13th ACM SIGKDD. pp. 420–429. ACM (2007)

[18] Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)

[19] Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N.S., Hurst, M.: Patterns of cascading behavior in large blog graphs. In: SDM. vol. 7, pp. 551–556. SIAM (2007)

[20] Liu, B., Cong, G., Xu, D., Zeng, Y.: Time constrained influence maximization in social networks. In: Data Mining (ICDM). pp. 439–448. IEEE (2012)

[21] Mohammadi, A., Saraee, M., Mirzaei, A.: Time-sensitive influence maximization in social networks. Journal of Information Science 41(6), 765–778 (2015)

[22] Prakash, B.A., Chakrabarti, D., Valler, N.C., Faloutsos, M., Faloutsos, C.: Threshold conditions for arbitrary cascade models on arbitrary networks. Knowledge and information systems 33(3), 549–575 (2012)

[23] Richardson, M., Domingos, P.: Mining knowledge-sharing sites for viral marketing. In: Proceedings of the eighth ACM SIGKDD. pp. 61–70. ACM (2002)

[24] Rodriguez, M.G., Schölkopf, B.: Influence maximization in continuous time diffusion networks. arXiv:1205.1682 (2012)

[25] Tang, Y., Shi, Y., Xiao, X.: Influence maximization in near-linear time: A martingale approach. In: Proceedings of the 2015 ACM SIGMOD. pp. 1539–1554. ACM (2015)

[26] Wu, H., Cheng, J., Huang, S., Ke, Y., Lu, Y., Xu, Y.: Path problems in temporal graphs. Proceedings of the VLDB Endowment 7(9), 721–732 (2014)