Tracking Influential Individuals in Dynamic Networks

Yu Yang, Zhefeng Wang, Jian Pei, *Fellow, IEEE*, and Enhong Chen, *Senior Member, IEEE*

Abstract—In this paper, we tackle a challenging problem inherent in a series of applications: tracking the influential nodes in dynamic networks. Specifically, we model a dynamic network as a stream of edge weight updates. This general model embraces many practical scenarios as special cases, such as edge and node insertions, deletions as well as evolving weighted graphs. Under the popularly adopted linear threshold model and independent cascade model, we consider two essential versions of the problem: finding the nodes whose influences passing a user specified threshold and finding the top-k most influential nodes. Our key idea is to use the polling-based methods and maintain a sample of random RR sets so that we can approximate the influence of nodes with provable quality guarantees. We develop an efficient algorithm that incrementally updates the sample random RR sets against network changes. We also design methods to determine the proper sample sizes for the two versions of the problem so that we can provide strong quality guarantees and, at the same time, be efficient in both space and time. In addition to the thorough theoretical results, our experimental results on five real network data sets clearly demonstrate the effectiveness and efficiency of our algorithms.

Index Terms—Social influence, dynamic networks

1 Introduction

More and more applications are built on dynamic networks and need to track influential nodes. For example, consider cold-start recommendation in a dynamic social network—we want to recommend to a new comer some existing users in a social network. A new user may want to subscribe to the posts from some users in order to obtain hot posts (posts that are widely spread in the social network) at the earliest time. Clearly for such a new user we should recommend her some influential users in the current network. Traditional Influence Maximization cannot find those influential users we want here because it is for marketing in which all seed users have to be synchronized to spread the same content, while in reality online influential individuals often produce and spread their own contents in an asynchronized manner. The influential users we want are those who have high individual influence.

More often than not, the underlying network is highly dynamic, where each node is a user and an edge captures the interaction from a user to another. User interactions evolve continuously over time. In an active social network, such as Twitter, Facebook, LinkedIn, Tencent WeChat, and Sina Weibo, the evolving dynamics, such as rich user interactions over time, is the most important value. It is critical to capture the most influential users in an online manner. To address the needs, we have to tackle two challenges at the same time, influence computation and dynamics in networks.

Influence computation is very costly, technically #P-hard under most influence models. Most existing studies have to compromise and consider the influence maximization problem only on a static network. Here, influence maximization in a network is to find a set of vertices S such that the combined influence of the nodes in the set is maximized and S satisfies some constraints such as the size of S is within a budget. The incapability of handling dynamics in large evolving networks seriously deprives many opportunities and potentials in applications. Also note that influence maximization is very different from finding influential individuals, for the reason that the best k-vertices set S does not consist of the k most influential individual nodes because influence spreads of different individuals may overlap.

Although influence maximization and finding influential nodes are highly related since they both need to compute influence in one way or another, these two problems serve very different application scenarios and face different technical challenges. For example, influence maximization is a core technique in viral marketing [1]. At the same time, influence maximization is not useful in the cold-start recommendation scenario discussed above, since a user is interested in being connected with individual users of great potential influence and may follow them in interaction.

To the best of our knowledge, our study is the first to tackle the problem of tracking influential nodes in dynamic networks. Please note that finding influential nodes is different from influence maximization. Specifically, we model a dynamic network as a stream of edge weight updates. Our model is general and embraces many practical scenarios as special cases. Under the popularly adopted linear threshold model and independent cascade model, we consider two essential versions of the problem: (1) finding the nodes whose influences passing a user specified threshold;

- *Y. Yang and J. Pei are with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. E-mail: yya119@sfu.ca, jpei@cs.sfu.ca.*
- *Z. Wang and E. Chen are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230000, China. E-mail: zhefwang@mail.ustc.edu.cn, cheneh@ustc.edu.cn.*

Manuscript received 7 Feb. 2017; revised 21 June 2017; accepted 17 July 2017. Date of publication 1 Aug. 2017; date of current version 4 Oct. 2017.
(Corresponding author: Yu Yang.)
Recommended for acceptance by Y. Zhang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2734667

and (2) finding the top-k most influential nodes. Our key idea is to use the polling-based methods and maintain a sample of random RR sets so that we can approximate the influence of nodes with provable quality guarantees.

Recently, there is encouraging progress in influence maximization on dynamic networks [2], [3], [4]. Due to the difference between influence maximization and finding influential nodes, the methods in those studies [2], [3], [4] cannot be applied directly to find influential nodes. Moreover, in terms of specific techniques, our study is also very different from [2], [3]. Most importantly, the methods in [2], [3] are heuristic, and do not provide any provable quality guarantee. Although authors of [4] claim that the algorithm in [4] has theoretical guarantees, in experiments reported, a key parameter is empirically set and makes the error rate ϵ even greater than 1. The reason that the algorithm in [4] cannot be implemented with small error rate is that the constant factor in its complexity is too large to be practical in use. In addition, the influence model considered in [2], [4] is the Independent Cascade model. The one in [3] is a non-linear system. We address both the Linear Threshold model and the Independent Cascade model in this study. To the best of our knowledge, we are the first to tackle influence computation with provable quality guarantee and report experiment results where algorithms are implemented strictly to fulfill the theoretical guarantee under the two most widely adopted influence models on dynamic networks.

To tackle the novel and challenging problem of finding influential nodes in dynamic networks, we make several technical contributions. We develop an efficient algorithm that incrementally updates the sample random RR sets against network changes. We also design methods to determine the proper sample sizes for the two versions of the problem so that we can provide strong quality guarantees and at the same time be efficient in both space and time. In addition to the thorough theoretical results, our experimental results on 5 real data sets clearly demonstrate the effectiveness and efficiency of our algorithms. The largest data set used contains over 41 million nodes, 1.5 billion edges and 0.3 billion edge updates.

The rest of the paper is organized as follows. We review the related work in Section 2. In Section 3, we recall the Linear Threshold model and the Independent cascade model, review the polling-based method for computing influence spread, and formulate influence in dynamic networks. In Section 4, we present methods updating random RR sets over a stream of edge weight updates. In Section 5, we tackle the problem of tracking nodes whose influence spreads pass a user-defined threshold. In Section 6, the problem of finding the top-k influential nodes is settled. We report the experimental results in Section 7. We conclude the paper in Section 8.

2 Related Work

Domingos et al. [1] proposed to take advantage of peer influence between users in social networks for marketing. Kempe et al. [5] formulated the problem using two discrete influence models, namely Independent Cascade model and Linear Threshold model. Since then, influence computation, especially influence maximization, has drawn much attention from both academia and industry [6], [7], [8], [9], [10], [11], [12], [13], [14]. Some heuristic methods were designed for computing influence spread under the Linear Threshold model [10], [11]. For the Independent Cascade Model, [12], [15] proposed approximations of influence spread estimations. Note that there are still gaps between estimations of influence spread and real influence spreads, which were not clearly quantified in [12], [15]. Consequently, both [15] and [12] cannot compute influence spread with provable quality guarantees. Recently, a polling-based method [8], [9], [16] was proposed for influence maximization under general triggering models. The key idea is to use some "Reversely Reachable" (RR) sets [9], [16] to approximate the real influence spread of nodes. The error of approximation can be bounded with a high probability if the number of RR sets is large enough.

Extracting influential nodes in social networks is also an important problem in social network analysis and has been extensively investigated [17], [18], [19], [20]. In addition to the marketing value, influential individuals are also useful in recommender systems in online web service [18], [19]. Due to the computational hardness of influence spread [10], [21], most methods did not use influence models to measure a user's influence, but adopted measures like PageRank which can be efficiently computed.

In a few applications, the underlying networks are evolving all the time [22], [23]. Rather than re-computing from scratch, incremental algorithms are more desirable in graph analysis tasks on dynamic networks. Maintaining PageRank values of nodes on an evolving graph was studied in [24], [25]. Hayashi et al. [26] proposed to utilize a sketch of all shortest paths to dynamically maintain the edge betweenness value. The dynamics considered by the above work is a stream of edge insertions/deletions, which is not suitable for influence computation. The dynamics of influence network is more complicated, because besides edge insertions/deletions, influence probabilities of edges may also evolve over time [27].

Aggarwal et al. [3] explored how to find a set of nodes that has the highest influence within a time window $[t_0, t_0 + h]$. They modeled influence propagation as a non-linear system which is very different from triggering models like the Linear Threshold model or the Independent Cascade model. The algorithm in [3] is heuristic and the results produced do not come with any provable quality guarantee.

Chen et al. [2] investigated incrementally updating the seed set for influence maximization under the Independent Cascade model. They proposed an algorithm which utilizes the seed set mined from the former network snapshot to efficiently find the seed set of the current snapshot. An Upper Bound Interchange heuristic is applied in the algorithm. However, the algorithm in [2] is costly in processing updates, since updating the Upper Bound vector for filtering non-influential nodes takes $O(m)$ time where m is the number of edges. Moreover, the SP1M heuristic [28], which does not have any approximation quality guarantee, was adopted in [2] for estimating influence spread of nodes. Thus, the set of influential nodes, even when the size of the seed set is set to 1, does not have any provable quality guarantee.

Independently and simultaneously[1] Ohsaka et al. [4] studied a related problem, maintaining a number of RR sets over a stream of network updates under the IC model such that $(1 - 1/e - \epsilon)$-approximation influence maximization queries can be achieved with probability at least $1 - \frac{1}{n}$. Our work is different from [4] in the following aspects. First, the

[1] Early versions of our paper can be found at https://arxiv.org/abs/1602.04490

TABLE 1
Frequently Used Notations

Notation	Description
$G = \langle V, E, w\rangle$	A social network, where each edge $(u, v) \in E$ is associated with an influence weight w_{uv}
w_{uv}	weight of the edge (u, v) (LT model); propagation probability of the edge (u, v) (IC model)
$n = \|V\|$	The number of nodes in G
$m = \|E\|$	The number of edges in G
$N^{in}(u)$	The set of in-neighbors of u
w_u	Self-weight of u
W_u	$W_u = w_u + \sum_{v \in N^{in}(u)} w_{vu}$, the total weight of u
p_{uv}	$p_{uv} = \frac{w_{uv}}{W_u}$, the probability that v is influenced by its neighbor u (LT Model)
I_u	The influence spread of node u
$\bar{I}$	The average influence spread of individual nodes
M	The number of random RR sets
$\mathcal{H}$	The hyper-graph consists of M random RR sets
$\mathcal{D}(u)$	The degree of $u \in V$ in $\mathcal{H}$
$\mathcal{F}_{\mathcal{R}}(u)$	$\mathcal{F}_{\mathcal{R}}(u) = \frac{\mathcal{D}(u)}{M}$, the fraction of random RR sets containing u
T	Influence threshold set by users
I_{max}	Influence spread of the most influential individual node
I^k	Influence spread of the $k-$th most influential individual node
${\mathcal{F}_{\mathcal{R}}}^*$	The highest $\mathcal{F}_{\mathcal{R}}(u)$ value for $u \in V$
${\mathcal{F}_{\mathcal{R}}}^k$	The $k-$th highest $\mathcal{F}_{\mathcal{R}}(u)$ value for $u \in V$

problems are different. The problem tackled in [4] is influence maximization, while our problem is tracking influential individuals. Second, [4] only studied the IC model while in our work we addressed both the IC and the LT models. Moreover, our algorithm is theoretically sound and was strictly implemented to fulfill the theoretical guarantee in experiments, while it is not the case in [4]. To enable theoretical guarantees for the algorithm in [4], one has to collect enough RR sets until the cost of all RR sets (i.e., the number of edges traversed when generating those RR sets) is $\Theta(\frac{(m+n)\log n}{\epsilon^3})$, which is a very large number in practice. Thus, in the experiments reported in [4], the demanded cost is empirically set to $32(m + n)\log n$, which means ϵ is even greater than 1, because the constant factor hidden in $\Theta(\frac{(m+n)\log n}{\epsilon^3})$ is greater than 32.

3 Preliminaries

In this section, we recall the Linear Threshold influence model and the Independent Cascade Model [5]. We also review the polling method for computing influence spread [8], [9], [16]. We then formulate influence in dynamic networks. For readers' convenience, Table 1 lists the frequently used notations.

3.1 Linear Threshold Model

Consider a directed social network $G = \langle V, E, w\rangle$ where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and each edge $(u, v) \in E$ is associated with an influence weight $w_{uv} \in [0, +\infty)$. Each node $v \in V$ also carries a weight w_v, which is called the *self-weight* of v. Denote by $W_v = w_v + \sum_{u \in N^{in}(v)} w_{uv}$ the total weight of v, where $N^{in}(v)$ is the set of v's in-neighbors.

We define the *influence probability* p_{uv} of an edge (u, v) as $\frac{w_{uv}}{W_v}$. Clearly, for $v \in V$, $\sum_{u \in N^{in}(v)} p_{uv} \leq 1$.

In the Linear Threshold (LT) model [5], given a seed set $S \subseteq V$, the influence propagates in G as follows. First, every node u randomly selects a threshold $\lambda_u \in [0, 1]$, which reflects our lack of knowledge about users' true thresholds. Then, influence propagates iteratively. Denote by S_i the set of nodes that are active in step i $(i = 0, 1, \ldots)$ and $S_0 = S$. In each step $i \geq 1$, an inactive node v becomes active if

$$\sum_{u \in N^{in}(v) \cap S_{i-1}} p_{uv} \geq \lambda_v.$$

The propagation stops at step t if $S_t = S_{t-1}$. Let $I(S)$ be the expected number of nodes that are finally active when the seed set is S. We call $I(S)$ the *influence spread* of S. Let I_u be the influence spread of a single node u.

Kempe et al. [5] proved that the LT model is equivalent to a "live-edge" process where each node v picks at most one incoming edge (u, v) with probability p_{uv}. Consequently, v does not pick any incoming edges with probability $1 - \sum_{u \in N^{in}(v)} p_{uv} = \frac{w_v}{W_v}$. All edges picked are "live" and the others are "dead". Then, the expected number of nodes reachable from $S \subseteq V$ through live edges is $I(S)$, the influence spread of S.

It is worth noting that our description of the LT model here is slightly different from the original [5]: we use a function of edge weights and self-weight of nodes to represent influence probabilities. Representing influence probabilities in this way is widely adopted in the existing literature [9], [10], [11], [16], [29].

3.2 Independent Cascade Model

A social network in the Independent Cascade (IC) model is also a weighted graph $G = \langle V, E, w\rangle$. Let w_{uv} represent the propagation probability of the edge (u, v), which is the probability that v is activated by u through the edge in the next step after u is activated. Clearly for the IC model, all $w_{uv} \in [0, 1]$.

In the IC model [5], given a seed set $S \subseteq V$, the influence propagates in G iteratively as follows. Denote by S_i the set of nodes that are active in step i $(i = 0, 1, \ldots)$ and $S_0 = S$. At step $i + 1$, each node u in S_i has a single chance to activate each inactive neighbor v with an independent probability w_{uv}. The propagation stops at step t if $S_t = \emptyset$. Similar to the LT model, the influence spread $I(S)$ denotes the expected number of nodes that are finally active when the seed set is S.

The "live-edge" process [5] of the IC model is to keep each edge (u, v) with a probability w_{uv} independently. All kept edges are "live" and the others are "dead". Then, the expected number of nodes reachable from S via live edges is the influence spread $I(S)$.

3.3 The Polling Method for Influence Computation

Computing influence spread is #P-hard under both the LT model and the IC model [10], [21]. Recently, a polling-based method [8], [9], [16] was proposed for approximating influence spread of triggering models [5] like the LT model and the IC model. Here we briefly review the polling method for computing influence spread.

Given a social network $G = \langle V, E, w\rangle$, a poll is conducted as follows: we pick a node $v \in V$ in random and then try to find out which nodes are likely to influence v. We run a Monte Carlo simulation of the equivalent "live-edge" process. The nodes that can reach v via live edges are considered as the potential influencers of v. The set of influencers found by each poll is called a *random RR (Reversely Reachable) set*.

Let $R_1, R_2, \ldots, R_M$ be a sequence of random RR sets generated by M polls, where M can also be a random variable. The M random RR sets form a random hyper-graph $\mathcal{H}$ where the set of nodes is still V and each random RR set is a hyper edge. Denote by $\mathcal{D}(S)$ the degree of a set of nodes S in the hyper-graph, which is the number of hyper-edges containing at least one node in S. Let $\mathcal{F}_{\mathcal{R}}(S) = \frac{\mathcal{D}(S)}{M}$. By the linearity of expectation, it has been shown that $n\mathcal{F}_{\mathcal{R}}(S)$ is an unbiased estimator of $I(S)$ [8], [9]. Tang et al. [9] proved that the corresponding sequence $x_1, x_2, \ldots, x_M$ is a martingale [30], where $x_i = 1$ if $S \cap RR_i \neq \emptyset$ and $x_i = 0$ otherwise. We have $E[\sum_{i=1}^{M} x_i] = E[\mathcal{D}(S)] = \frac{MI(S)}{n}$. The following results [9] show how $E[\sum_{i=1}^{M} x_i]$ is concentrated around $\frac{MI(S)}{n}$.

Corollary 1 ([9]). *For any $\xi > 0$,*

$$\Pr\Big[\sum_{i=1}^{M} x_i - Mp \geq \xi Mp\Big] \leq \exp\Big(-\frac{\xi^2}{2+\frac{2}{3}\xi}Mp\Big)$$

$$\Pr\Big[\sum_{i=1}^{M} x_i - Mp \leq -\xi Mp\Big] \leq \exp\Big(-\frac{\xi^2}{2}Mp\Big),$$

where $p = \frac{I(S)}{n}$.

Sections 5 and 6 will use the above results to analyze how many random RR sets are needed for extracting influential nodes. Note that since the problem we study in this paper is different from influence maximization, the results (theorems and lemmas) in [9] cannot be applied to our analysis.

3.4 Influence in Dynamic Networks

Real online social networks, such as the Facebook network and the Twitter network, change very fast and all the time. Relationships among users keep changing, and influence strength of relationships also varies over time. Lei et al. [27] pointed out that influence probabilities may change due to former inaccurate estimation or evolution of users' relations over time. However, the traditional formulation of dynamic networks only considers the topological updates, that is, edge insertions and edge deletions [24], [25], [26]. Such a formulation is not suitable for realtime accurate analysis of influence.

According to the LT model reviewed in Section 3.1, the change of influence probabilities along edges can be reflected by the change of edge weights. For the IC model, since the weight of an edge is the propagation probability, the updates on edge weights are updates on propagation probabilities. Therefore, we model a dynamic network as a stream of weight updates on edges.

A *weight update* on an edge is a 5-tuple $(u, v, +/-, \Delta, t)$, where (u, v) is the edge updated, $+/-$ is a flag indicating whether the weight of (u, v) is increased or decreased, $\Delta > 0$ is the amount of change to the weight and t is the time stamp. The update is applied to the self-weight w_u if $u = v$. Clearly, edge insertions/deletions considered in the existing literature [2], [24], [25], [26] can be easily written as weight increase/decrease updates. Moreover, node insertions/deletions can be written as edge insertions/deletions, too.

Example 1. A retweet network is a weighted graph $G = \langle V, E, w\rangle$, where V is a set of users. An edge $(u, v) \in E$ captures that user v retweeted from user u. We can set w_{uv} according to the propagation model adopted as follows.

LT Model: The edge weight w_{uv} is the number of tweets that v retweeted from u. The self-weight w_v is the number of original tweets posted by v. The weights reflect the influence in the social network. By intuition, if v retweeted many tweets from u, v is likely to be influenced by u. In contrast, if most of v's tweets are original, v is not likely to be influenced by others.

IC Model: The edge weight w_{uv} is the probability that v retweets from u, which can be calculated according to v's retweeting record in the past [29], [31].

An essential task in online social influence analysis is to capture how the influence changes over time. For example, one may want to consider only the retweets within the past Δt time. Clearly, the set of edges E may change and the weights w_{uv} and w_v may increase or decrease over time. The dynamics of the retweet network can be depicted by a stream of edge weight updates $\{(u, v, +/-, \Delta, t)\}$.

Given a dynamic network like the retweet network in Example 1, how can we keep track of influential users dynamically? In order to know the influential nodes, the critical point is to monitor influence of users. To solve this problem, we adopt the polling-based method for computing influence spread, and extend it to tackle dynamic networks. The major challenge is how to maintain a number of RR sets over a stream of weight updates, such that $n\mathcal{F}_{\mathcal{R}}(S)$ is always an unbiased estimator of $I(S)$. We propose a framework for updating RR sets that addresses various tasks of tracking influential nodes.

The framework is shown in Algorithm 1. In Section 4, we discuss how to efficiently update the existing RR sets. How to decide if our current RR sets are insufficient, redundant or in proper amount depends on the specific task of tracking influential nodes. In Sections 5 and 6, respectively, we discuss this issue for two common tasks of tracking influential nodes, namely tracking nodes with influence greater than a threshold and tracking top-k influential nodes.

4 Updating RR Sets

In this section, we propose an incremental algorithm for updating existing RR sets over a stream of edge weight updates under both the LT model and the IC model. We prove that, by updating RR sets using our algorithm, $n\mathcal{F}_{\mathcal{R}}(S)$ is always an unbiased estimation of $I(S)$. We also analyze the cost of an update based on the assumption that we are maintaining in total M RR sets. Note that the value of M should be decided for specific tasks. In Sections 5 and 6 we discuss the value of M for two common tasks of tracking influential nodes.

Algorithm 1. Framework of Updating RR Sets

```
retrieve RR Sets affected by the updates of the graph
update retrieved RR sets
if the current RR sets are insufficient then
  add new RR sets
else
  if the current RR sets are redundant then
    delete the redundant RR sets
  end if
end if
```

4.1 Updating Under the LT Model

First, we have a key observation about random RR sets for the LT model.

Fact 1. *A random RR set of the LT model is a simple path.*

Rationale. In the equivalent "live-edge" selection process of the LT model, each node selects at most one

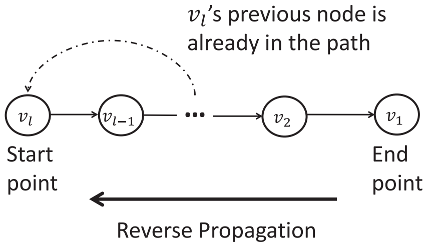


Fig. 1. A random path. v_i is the previous node of v_{i-1}.

incoming edge as a live edge. In the polling process, a random RR set is the set of nodes that can be reversely reachable from a randomly picked node v via live edges. Thus, the nodes in a random RR set together form a simple path.

Fig. 1 illustrates a random RR set. The end point v_1 is picked in random at the beginning of the polling process. Then the path is generated by reversely propagating from v_1. The reverse propagation ends at v_l because v_l picks one of the nodes already in the path as its previous node. Note that the situation that v_l does not pick any previous nodes can be regarded as v_l picks itself as the previous node.

For a random RR set, suppose the starting node is v_l, we also store the previous node picked by v_l, which is useful in our algorithm for updating random RR sets maintained. Clearly the space complexity of a RR set is $O(L)$ where L is the number of nodes in the RR set. We maintain an inverted index on all random RR sets so that we can access all the random RR sets passing a node. Moreover, we assume that the whole graph is stored and maintained in a way allowing random access to every node and its in-neighbors. It is not difficult to verify that the expected number of nodes of a RR set is $\bar{I}$, the average individual influence in the network. Thus, the expected space cost of M RR sets and the inverted index is $O(M\bar{I} + n)$.

When there is an edge weight update $(u, v, +/-, \Delta, t)$ at time t, our incremental algorithm works as follows. Denote by w_{uv}^t the edge weight of (u, v) and W_v^t the total weight of v at time t. We first update the edge weight of (u, v) and the total weight of v in the graph. Then, we consider the following two cases.

1) If the update is a weight increase $(u, v, +, \Delta, t)$, we retrieve all RR sets passing v using the inverted index. For each RR set retrieved, with probability $\frac{\Delta}{W_v^t}$ it is rerouted from v. If a RR set is rerouted, the previous node of v is set to u and we keep reversely propagating until no new nodes can be reversely reached.
2) If the update is a weight decrease $(u, v, -, \Delta, t)$, we retrieve all RR sets passing v where the previous node of v is u. Each retrieved RR set is rerouted from v with probability $\frac{\Delta}{w_{uv}^{t-1}}$. If a RR set is rerouted, we choose u' among the in-neighbors of v at time t as the previous node of v with probability $\frac{w_{u'v}^t}{W_v^t}$. We keep reversely propagating until no new nodes can be reversely reached.

When rerouting random RR sets, we use random access to obtain the nodes and the in-neighbors of them in the graph. We also update the inverted index.

The update operations are similar to Reservoir Sampling [32]. It is easy to prove that, at any time t, after the incremental maintenance, for any (u, v) where u is an in-neighbor of v, u is picked as the previous node of v with probability $\frac{w_{uv}^t}{W_v^t}$. Thus, $n\mathcal{F}_{\mathcal{R}}(S)$ is always an unbiased estimator of $I(S)$ for any S.

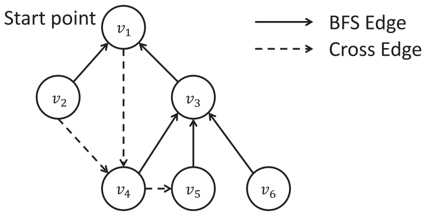


Fig. 2. A random RR set of the IC model is a random connected component.

Theorem 1. *At any time t, after our incremental maintenance of the random RR sets under the LT model as described in this section, $n\mathcal{F}_{\mathcal{R}}(S)$ is an unbiased estimator of $I(S)$ for any seed set S.*

Limited by space, we skip the proof which can be found at an early version of this work.[2]

The expected number of RR sets needed to be retrieved is $\frac{MI_v^{t-1}}{n} \ll M$ for an update $(u, v, +/-, \Delta, t)$. Only a small fraction of the retrieved RR sets need to be updated. Specifically, the expected number of RR sets updated is $\frac{MI_v^{t-1}\Delta}{nW_v^t} \ll M$ for a weight increase update $(u, v, +, \Delta, t)$, and $\frac{MI_v^{t-1}\Delta}{nW_v^{t-1}} \ll M$ for a weight decrease update $(u, v, -, \Delta, t)$. Clearly the cost of incremental maintenance is much less than re-generating M RR sets from scratch.

4.2 Updating Under the IC Model

The idea of updating RR sets under the IC model is similar to [4]. We briefly introduce the idea in this section.

Rather than a simple path, a random RR set in the IC model is a random connected component. Fig. 2 illustrates an example. Suppose the start point (the randomly picked node at the beginning of a poll) of a RR set is v_1, then each node in this RR set can be reversely reachable from v_1 via live edges.

For a random RR set, we not only record the nodes in it but also all live edges among those nodes. We categorize live edges into two classes, namely BFS edges and cross edges. When a RR set is being generated by reversely propagating from the start point in a breadth-first search manner, if a live edge (v_i, v_j) makes v_i propagated for the first time, (v_i, v_j) is labeled as a BFS edge; otherwise it is labeled as a cross edge. For each node in a RR set, we use an adjacent list to store all live edges pointing to it. We also treat every node as a string and keep all nodes in a RR set in a prefix tree for fast retrieving a node and the address of its adjacent list of live edges. The major difference of our data structure for storing a RR set to the one in [4] is we do not store the propagation probabilities on live edges in a RR set, while [4] does. We only store propagation probabilities in the graph data structure. This is obviously an improvement in space because the propagation probability of an edge is only stored once in our method.

Like the LT model, for the IC model, we also maintain an inverted index on all random RR sets so that we can access all RR sets containing a node. Since in the "live-edge" process of the IC model, every edge is picked independently, when there is an update $(u, v, +/-, \Delta, t)$ at time t, status ("live" or "dead") of edges other than (u, v) in RR sets stay the same. Thus, we have the following incremental maintenance,

[2] https://arxiv.org/abs/1602.04490

1) If the update is a weight increase $(u, v, +, \Delta, t)$, we retrieve all RR sets passing v using the inverted index. For each RR set retrieved, if (u, v) is not a live edge of it, we add (u, v) as a live edge to it with probability $\frac{\Delta}{1-w_{uv}^{t-1}}$. After adding (u, v), if u does not belong to this RR set at time $t-1$, we further extend this RR set by reversely propagating from u in a breadth-first search manner.
2) If the update is a weight decrease $(u, v, -, \Delta, t)$, we retrieve all RR sets passing v. If a retrieved RR set contains a live edge (u, v), with probability $\frac{\Delta}{w_{uv}^{t-1}}$ we remove (u, v). If (u, v) is removed, we traverse from the start point v_1 via live edges other than (u, v) of this RR set to find all nodes reversely reachable from v_1 and all live edges among them. Then, this RR set is updated to one containing only those nodes and live edges we find.

Similar to the LT model, after updating the RR sets, we also update the inverted index.

Clearly, our incremental maintenance ensures that, for each edge (u, v) at time t, if v is a node of a RR set, the probability that (u, v) is a live edge of this RR set is w_{uv}^t. So the same as the LT model, our incremental maintenance ensures that from the RR sets we can always have unbiased estimations of influence spreads.

Theorem 2. *At any time t, after our incremental maintenance of the random RR sets under the IC model as described in this section, $n\mathcal{F}_{\mathcal{R}}(S)$ is an unbiased estimator of $I(S)$ for any seed set S.*

In our incremental maintenance, we need to find out if an edge (u, v) is a live edge in a RR set. Suppose the number of nodes in a RR set is L. Because normally the length of a node id is a constant, given an edge (u, v), using the prefix tree we can find the address of v's adjacent list in $O(1)$ time. Then a linear search is performed to find out if (u, v) is a live edge. In practice propagation probabilities are often small and $\sum_{u \in N^{in}(u)} w_{uv}$ is often a small constant. Therefore, in practice the average complexity of the linear search is $O(1)$ and in total we only need $O(1)$ time to decide if (u, v) is a live edge in a RR set. Moreover, the space complexity of the RR set is $O(L)$ in practice since every node only has a constant number of live edges pointing to it. Similar to the LT model, maintaining M RR sets and the inverted index under the IC model takes $O(M\bar{I} + n)$ space in expectation, where $\bar{I}$ is the average individual influence.

For the second situation when a live edge (u, v) is deleted, it is not always necessary to traverse from the start point, which takes $O(L)$ time if there are L nodes in the RR set. It is easy to see that removing cross edges does not change the connectivity of nodes in a RR set. Thus, if the removed live edge is labeled as a cross edge, we do not need to further update the RR set.

Similar to LT model, under IC model, the expected number of RR sets needed to be retrieved is $\frac{MI_v^{t-1}}{n} \ll M$ for an update $(u, v, +/-, \Delta, t)$ and only a small fraction of the retrieved RR sets need to be updated. The expected number of RR sets containing a live edge (u, v) is $\frac{MI_v^{t-1} w_{uv}^{t-1}}{n}$ and the expected number of RR sets that do not contain (u, v) as a live edge is $\frac{MI_v^{t-1}(1-w_{uv}^{t-1})}{n}$. Therefore, when there is an update on the edge (u, v), no matter it is weight increase or weight decrease, the expected number of RR sets needed to be updated is $\frac{MI_v^{t-1}\Delta}{n} \ll M$. Clearly the cost of incremental maintenance is much less than re-generating M RR sets from scratch.

5 Tracking Threshold-Based Influential Nodes

A natural problem setting of finding influential nodes is to find all nodes whose influence spread is at least T, where T is a user-specified threshold. In this section, we discuss how to use random RR sets to approximate the desired result.

Before our discussion, we clarify that our problem is not Heavy Hitters [33] even when we treat the influence spread of a node as the "frequency/popularity" of an element. First, the definitions of "frequency" are different and have dramatically different properties. In Heavy Hitters, a stream of items is a multiset of elements and the frequency of an element is its multiplicity over the total number of items. Thus, the sum of frequencies of all elements is 1, which means there are at most $1/\phi$ elements with frequency passing a threshold ϕ. In our problem, if we define the "frequency" of a node v as I_v/n, the value of $\sum_{v \in V} \frac{I_v}{n}$ is not necessarily 1. Actually one can easily prove that computing $\sum_{v \in V} I_v$ is #P-hard because computing I_v is #P-hard. As a result, normalizing I_v is difficult. Thus, given any influence threshold $T < n$, we cannot have an upper bound on the number of nodes that have influence greater than T. Also, the input of our problem is a stream of edge updates but not a stream of insertion/deletion of nodes (elements). Moreover, the influence of a node is not a simple aggregation of weights on the associated edges. In terms of technical solutions, it is hard to use a sublinear space to convert an update of edge weight to a list of insertions/deletions of nodes. As illustrated in Section 4, we need both the graph and RR sets to decide which nodes should be increased/decreased in frequency by an edge update. This is very different from the settings of Heavy Hitters where only a sublinear space is allowed, while the graph itself already takes space $\Omega(n)$. We also need to access a number of RR sets, while in Heavy Hitters only counters of elements are allowed to be kept in memory.

Due to the #P-hardness of computing influence spread under the LT model [10], it is not likely that we can find in polynomial time the exact set of nodes whose influence spread is at least T. Thus, we turn to algorithms that allow controllable small errors. Specifically, we ensure that the recall of the set of nodes found by our algorithm is 100 percent and we tolerate some false positive nodes. Moreover, the influence spread of those false positive nodes should take a high probability to have a lower bound that is not much smaller than T. We set the lower bound to $T - \epsilon n$, where ϵ controls the error.

According to Corollary 1, the larger M, the more accurate the unbiased estimator $n\mathcal{F}_{\mathcal{R}}(u)$. Thus, the intuition of deciding M is to make sure that, for each u, $n\mathcal{F}_{\mathcal{R}}(u)$ is large enough when $I_u \geq T$, and small enough when $I_u \leq T - \epsilon n$.

We first show that $n\mathcal{F}_{\mathcal{R}}(u)$ is not likely to be too much smaller than T if $I_u \geq T$ and M is large enough.

Lemma 1. *With M random RR sets, if $I_u \geq T$, with probability at least $1 - \exp(-\frac{M\epsilon^2 n}{8T})$, $n\mathcal{F}_{\mathcal{R}}(u) \geq T - \frac{\epsilon n}{2}$.*

Proof. If $I_u \geq T$, we have

$$\begin{aligned}\Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq T - \frac{\epsilon n}{2}\Big\} &= \Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq I_u - \Big(I_u - T + \frac{\epsilon n}{2}\Big)\Big\} \\ &= \Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq \Big(1 - \frac{I_u - T + \frac{\epsilon n}{2}}{I_u}\Big)I_u\Big\} \\ &\leq \exp\Big\{-\frac{M(I_u - T + \frac{\epsilon n}{2})^2}{2nI_u}\Big\}\end{aligned}$$

$\frac{(I_u - T + \frac{\epsilon n}{2})^2}{I_u}$ is non-decreasing with respect to I_u when $I_u \geq T$. Thus,

$$\Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq T - \frac{\epsilon n}{2}\Big\} = \exp\Big(-\frac{M\epsilon^2 n}{8T}\Big).$$

□

Similarly, if $I_u \leq T - \epsilon n$, the probability that $n\mathcal{F}_{\mathcal{R}}(u)$ is abnormally large is pretty small when M is large.

Lemma 2. *With M random RR sets, if $I_u \leq T - \epsilon$, with probability at least $1 - 2\exp(-\frac{M\epsilon^2 n}{12T})$, $n\mathcal{F}_{\mathcal{R}}(u) \leq T - \frac{\epsilon n}{2}$.*

Proof. We prove that if $I_u \leq T - \epsilon n$, $\Pr\{n\mathcal{F}_{\mathcal{R}}(u) - I_u \geq \frac{\epsilon n}{2}\} \leq 2\exp(-\frac{M\epsilon^2 n}{12T})$. Note that $n\mathcal{F}_{\mathcal{R}}(u) - I_u \leq \frac{\epsilon n}{2}$ is a sufficient condition for $n\mathcal{F}_{\mathcal{R}}(u) \leq T - \frac{\epsilon n}{2}$ when $I_u \leq T - \epsilon n$.

First, suppose $T \geq \frac{3\epsilon n}{2}$, which means $\frac{\epsilon n}{2} \leq T - \epsilon n$. There are two possible cases.

Case 1. $\frac{\epsilon n}{2} \leq I_u \leq T - \epsilon n$. Then,

$$\begin{aligned}
&\Pr\Big\{|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \geq \frac{\epsilon n}{2}\Big\} \\
&= \Pr\Big\{|M\mathcal{F}_{\mathcal{R}}(u) - \frac{MI_u}{n}| \geq \frac{\epsilon M}{2}\Big\} \\
&\leq 2\exp\Big\{-\frac{1}{3}\frac{MI_u}{n}\frac{\epsilon^2 n^2}{4I_u^2}\Big\} \leq 2\exp\Big(-\frac{M\epsilon^2 n}{12T}\Big).
\end{aligned}$$

Case 2. $I_u \leq \frac{\epsilon n}{2}$. Then,

$$\begin{aligned}
&\Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) - I_u \geq \frac{\epsilon n}{2}\Big\} \\
&= \Pr\Big\{M\mathcal{F}_{\mathcal{R}}(u) - \frac{MI_u}{n} \geq \frac{\epsilon M}{2}\Big\} \\
&\leq \exp\Big\{-\frac{1}{(2+\frac{2}{3})\frac{\epsilon n}{2I_u}}\frac{MI_u}{n}\frac{\epsilon^2 n^2}{4I_u^2}\Big\} \\
&\leq \exp\Big\{-\frac{3M\epsilon}{16}\Big\} \leq 2\exp\Big(-\frac{M\epsilon^2 n}{12T}\Big).
\end{aligned}$$

Second, if $T \leq \frac{3\epsilon n}{2}$, for all $I_u \leq T - \epsilon n$, $I_u \leq \frac{\epsilon n}{2}$. Then, all $I_u \leq T - \epsilon n$ fall into Case 2 above and the lemma still holds. □

Because $\exp(-\frac{M\epsilon^2 n}{8T}) \leq 2\exp(-\frac{M\epsilon^2 n}{12T})$, by applying Boole's inequality (that is, the Union Bound), with probability at least $1 - 2n \cdot \exp(-\frac{M\epsilon^2 n}{12T})$, every $n\mathcal{F}_{\mathcal{R}}$ satisfies the conditions in Lemmas 1 and 2. Therefore, we have the following theorem on the sample size M for finding nodes whose influence spread is at least T.

Theorem 3. *By setting the number of random RR sets $M = \frac{12T}{n\epsilon^2}\ln\frac{2n}{\delta}$, with probability at least $1 - \delta$ the following conditions hold for every node u.*

1) *If $I_u \geq T$, then $n\mathcal{F}_{\mathcal{R}}(u) \geq T - \frac{\epsilon n}{2}$*
2) *If $I_u < T - \epsilon n$, then $n\mathcal{F}_{\mathcal{R}}(u) < T - \frac{\epsilon n}{2}$*

One nice property of M in Theorem 3 is that, given n, T, ϵ and δ, M is a constant. Therefore, when we track nodes of influence spread at least T in a dynamic network, no matter how the network changes, the sample size M remains the same.

6 Tracking Top-K Most Influential Nodes

Another useful problem setting is to find the top-k influential nodes, where k is a user-specified parameter.

Denote by I^k the influence spread of the kth most influential node. Extracting top-k influential individual nodes equals extracting all nodes whose influence spread is at least I^k. Again, due to the #P-hardness of influence computation, we probably have to tolerate errors in the result when designing algorithms. Similar to the task in Section 5, we hope the result returned by our algorithm contains all real top-k nodes, and for each false-positive node returned, its influence spread is no smaller than $I^k - \epsilon n$ with a high probability.

In this section, we first analyze the number of random RR sets M we need to achieve the above goal with a high probability. We show that M is proportional to the maximum individual influence spread I_{max} and devise an algorithm that can give a really good estimation of I_{max} with a high probability. Then, combining the theoretical results in Section 5, we propose a method that improves the precision of the result set of nodes, that is, reducing the number of false-positive nodes.

6.1 Sample Size

Unlike the task in Section 5, we do not know the threshold I^k in advance. Thus when selecting nodes according to values of $n\mathcal{F}_{\mathcal{R}}(u)$, we do not have a threshold value. This is similar to mining top-k itemsets using sampled transactions [34], [35]. The intuition of our idea to solve the problem is that, if we have enough samples, we can bound the threshold value within a small range.

To collect all real top-k influential nodes and filter out all nodes whose influence spreads are smaller than $I^k - \epsilon n$, we sample enough random RR sets such that for every $u \in V$, $|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \leq \frac{\epsilon n}{4}$ with a high probability. Denote by ${\mathcal{F}_{\mathcal{R}}}^k$ the kth highest $\mathcal{F}_{\mathcal{R}}$ value. We have the following result.

Lemma 3. *If for all $u \in V$, $|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \leq \frac{\epsilon n}{4}$, then the following conditions hold. (1) if $I_u \geq I^k$, then $n\mathcal{F}_{\mathcal{R}}(u) \geq n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{2}$; and (2) if $I_u \leq I^k - \epsilon n$, then $n\mathcal{F}_{\mathcal{R}}(u) \leq n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{2}$.*

Proof. First, $n{\mathcal{F}_{\mathcal{R}}}^k \geq I^k - \frac{\epsilon n}{4}$ because there are at least k nodes having the value of $n\mathcal{F}_{\mathcal{R}}$ at least $I^k - \frac{\epsilon n}{4}$. Second, $n{\mathcal{F}_{\mathcal{R}}}^k \leq I^k + \frac{\epsilon n}{4}$ because there are at most k nodes having the value of $n\mathcal{F}_{\mathcal{R}}$ at least $I^k + \frac{\epsilon n}{4}$. Thus, we have $n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{4} \leq I^k \leq n{\mathcal{F}_{\mathcal{R}}}^k + \frac{\epsilon n}{4}$.

If $I_u \geq I^k$, we have $n\mathcal{F}_{\mathcal{R}}(u) \geq I_u - \frac{\epsilon n}{4} \geq I^k - \frac{\epsilon n}{4} \geq n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{2}$.

If $I_u \leq I^k - \epsilon n$, we have $n\mathcal{F}_{\mathcal{R}}(u) \leq I_u + \frac{\epsilon n}{4} \leq I^k - \frac{3\epsilon n}{4} \leq n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{2}$. □

So we need to derive a lower bound of M to make sure $|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \leq \frac{\epsilon n}{4}$ for every $u \in V$ with a high probability. Denote by I_{max} the maximum individual influence spread.

Lemma 4. *When the number of random RR sets is M, with probability at least $1 - 2\exp(-\frac{Mn\epsilon^2}{48I_{max}})$, $|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \leq \frac{\epsilon n}{4}$.*

Proof. We need to consider two possible cases.

Case 1. If $I_u \geq \frac{\epsilon n}{4}$

$$\begin{aligned}
\Pr\Big\{|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \geq \frac{\epsilon n}{4}\Big\} &\leq 2\exp\Big\{-\frac{1}{3}\frac{\epsilon^2 n^2}{16I_u^2}\frac{MI_u}{n}\Big\} \\
&\leq 2\exp\Big(-\frac{Mn\epsilon^2}{48I_{max}}\Big).
\end{aligned}$$

Case 2. If $I_u \leq \frac{\epsilon n}{4}$

$$\begin{aligned}&\Pr\Big\{|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \geq \frac{\epsilon n}{4}\Big\}\\ &= \Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) - I_u \geq \frac{\epsilon n}{4}\Big\} \leq \exp\Big(-\frac{3}{8}\frac{\epsilon n}{4I_u}\frac{MI_u}{n}\Big)\\ &= \exp\Big(-\frac{3M\epsilon}{32}\Big) \leq 2\exp\Big(-\frac{Mn\epsilon^2}{48I_{max}}\Big). \qquad \square\end{aligned}$$

By applying the Union Bound, with probability at least $1 - 2n \cdot \exp(-\frac{Mn\epsilon^2}{48I_{max}})$, we have $|n\mathcal{F}_{\mathcal{R}}(u) - I_u| \leq \frac{\epsilon n}{4}$ for $u \in V$. In sequel, we have the following theorem settling the value of M.

Theorem 4. *By setting the number of random RR sets $M = \frac{48I_{max}}{n\epsilon^2}\ln\frac{2n}{\delta}$, with probability at least $1-\delta$ the following conditions hold. (1) If $I_u \geq I^k$, then $n\mathcal{F}_{\mathcal{R}}(u) \geq n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{2}$; and (2) If $I_u < I^k - \epsilon n$, then $n\mathcal{F}_{\mathcal{R}}(u) < n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{2}$.*

Algorithm 2. Sampling Sufficient Random RR sets for Top-K Influential Individuals

```
Input: $G = \langle V, E, w\rangle$, $\epsilon$, $\delta$ and $\mathcal{R}$ which is a set of random RR sets
Output: $\mathcal{R}$
1: while $|\mathcal{R}| < \frac{48\times 4\epsilon}{\epsilon^2}\ln\frac{2n}{\delta}$ do
2:     Sample a random RR set and add to $\mathcal{R}$
3: end while
4: $x \leftarrow \frac{|\mathcal{R}|\epsilon^2}{48\ln\frac{2n}{\delta}}$
5:     while ${\mathcal{F}_{\mathcal{R}}}^* \geq x - \epsilon$ do
6:     Sample a random RR sets and add to $\mathcal{R}$
7:     $x \leftarrow \frac{|\mathcal{R}|\epsilon^2}{48\ln\frac{2n}{\delta}}$
8: end while
9: return $\mathcal{R}$
```

Unlike [34], [35], [36], the sample size in Theorem 4 not only depends on the confidence level $1-\delta$ and the error ϵ, but also is proportional to I_{max}, which varies over different datasets. This is meaningful in practice, because for a social network, I_{max} is normally very small comparing to n [7], [9], [10], [11], [29], [37]. One may link finding influential nodes with finding frequent itemsets remotely due to the intuition that a node frequent in many RR sets is likely influential. In sampling based frequent itemsets mining [34], [35], [36], the sample size is decided by ϵ and δ only, and thus is in general larger than ours here.

6.2 Estimating I_{max} while Sampling

The sample size M in Theorem 4 depends on I_{max}, which is unknown and hard to compute in exact. In this section, we devise a sampling algorithm that gives a tight upper bound of I_{max} with a high probability and at the same time samples enough random RR sets we need.

Our algorithm sets $M = \frac{48x}{\epsilon^2}\ln\frac{2n}{\delta}$ and progressively increases xn until it is enough larger than I_{max}. The intuition is that, if $n{\mathcal{F}_{\mathcal{R}}}^*$ (${\mathcal{F}_{\mathcal{R}}}^*$ is the highest $\mathcal{F}_{\mathcal{R}}(u)$ value) is sufficiently smaller than xn, probably the current M is large enough.

Algorithm 2 shows our sampling method. We prove that the final random RR sets are enough and xn, the upper bound of I_{max}, is tight.

Lemma 5. *When $M = \frac{48x}{\epsilon^2}\ln\frac{2n}{\delta}$, if $I_{max} \geq xn$, with probability at least $1-\delta_1$, ${\mathcal{F}_{\mathcal{R}}}^* \geq x - \epsilon$, where $\delta_1 = (\frac{\delta}{2n})^{24}$ and ${\mathcal{F}_{\mathcal{R}}}^*$ is the maximum $\mathcal{F}_{\mathcal{R}}(u)$ for all $u \in V$.*

Proof. Suppose u is a node with the maximum influence. Since $xn \leq I_{max}$, we have

$$\begin{aligned}\Pr\{{\mathcal{F}_{\mathcal{R}}}^* \leq x - \epsilon\} &\leq \Pr\{\mathcal{F}_{\mathcal{R}}(u) \leq x - \epsilon\}\\ &= \Pr\Big\{\mathcal{F}_{\mathcal{R}}(u) \leq \Big(1 - \frac{\epsilon}{x}\Big)x\Big\}\\ &\leq \Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq \Big(1 - \frac{\epsilon}{x}\Big)I_{max}\Big\}\\ &\leq \exp\Big(-\frac{(\frac{\epsilon}{x})^2 MI_{max}}{2n}\Big)\\ &\leq \exp\Big(-\frac{(\frac{\epsilon}{x})^2 Mx}{2}\Big) = \Big(\frac{\delta}{2n}\Big)^{24}. \qquad \square\end{aligned}$$

Lemma 5 shows that with a high probability, if ${\mathcal{F}_{\mathcal{R}}}^* < x - \epsilon$, then the current random RR sets are enough.

Lemma 6. *When $xn \geq I_{max}$, $M = \frac{48x}{\epsilon^2}\ln\frac{2n}{\delta}$, with probability at least $1-\delta_2$, $\forall u$, if $I_u \leq (x-2\epsilon)n$, then $\mathcal{F}_{\mathcal{R}}(u) \leq x - \epsilon$, where $\delta_2 = n(\frac{\delta}{2n})^{16}$.*

Proof. Let $\epsilon' = \frac{\epsilon}{x}$. Note that the first 3 lines of Algorithm 2 ensure that $x \geq 4\epsilon$ and $\epsilon' \leq \frac{1}{4}$. Thus, $\frac{1-\epsilon'}{2} \leq 1 - 2\epsilon'$. We have two possible cases.

Case 1. If $\frac{(1-\epsilon')xn}{2} \leq I_u \leq (1-2\epsilon')xn$

$$\begin{aligned}&\Pr\{\mathcal{F}_{\mathcal{R}}(u) \geq x - \epsilon\}\\ &= \Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq \Big[1 + \frac{(1-\epsilon')xn - I_u}{I_u}\Big]I_u\Big\}\\ &\leq \exp\Big(-\frac{1}{3}\frac{[(1-\epsilon')xn - I_u]^2}{I_u^2}\frac{MI_u}{n}\Big)\\ &\leq \exp\Big(-\frac{\epsilon'^2 Mx}{3(1-2\epsilon')}\Big) \leq \exp\Big(-16\ln\frac{2n}{\delta}\Big) = \Big(\frac{\delta}{2n}\Big)^{16}.\end{aligned}$$

Case 2. If $I_u \leq \frac{(1-\epsilon')xn}{2}$

$$\begin{aligned}&\Pr\{n\mathcal{F}_{\mathcal{R}}(u) \geq (1-\epsilon')xn\}\\ &= \Pr\Big\{n\mathcal{F}_{\mathcal{R}}(u) \leq \Big[1 + \frac{(1-\epsilon')xn - I_u}{I_u}\Big]I_u\Big\}\\ &\leq \exp\Big(-\frac{3}{8}\frac{[(1-\epsilon')xn - I_u]}{I_u}\frac{MI_u}{n}\Big)\\ &\leq \exp\Big(-\frac{3(1-\epsilon')Mx}{16}\Big)\\ &\leq \exp\Big(\frac{9\ln\frac{2n}{\delta}}{2\epsilon'^2}\Big) \leq \exp\Big(-18\ln\frac{2n}{\delta}\Big) = \Big(\frac{\delta}{2n}\Big)^{18}.\end{aligned}$$

Applying the Union Bound, we have that, with probability at least $1 - n(\frac{\delta}{2n})^{16}$, $\mathcal{F}_{\mathcal{R}}(u) \leq x - \epsilon$ for any u such that $I_u \leq (x-2\epsilon)n$. □

Lemma 6 implies that when $(x-2\epsilon)n \geq I_{max}$, ${\mathcal{F}_{\mathcal{R}}}^* \leq x - \epsilon$ with a high probability. The first time in Algorithm 2 when $(x-2\epsilon)n \geq I_{max}$ we have $xn \leq \max(4\epsilon n, I_{max} + 2\epsilon n)$. If we set ϵ smaller than $\frac{I_{max}}{2n}$), the upper bound xn is at most $2I_{max}$. This is achievable in practice since I_{max} has some trivial lower bounds, such as $\max_{u\in V}\sum_{v\in N^{out}(u)} p_{uv}$.

Theorem 5. *Given ϵ and δ, with probability $1 - o(\frac{1}{n^{14}})$, Algorithm 2 returns $M = \frac{48x}{\epsilon^2}\ln\frac{2n}{\delta} \geq \frac{48I_{max}}{n\epsilon^2}\ln\frac{\delta}{2n}$ random RR sets, and $xn \leq \max(4\epsilon n, I_{max} + 2\epsilon n)$.*

Proof. There are only two possible reasons that Algorithm 2 may fail to achieve the above goals: (1) it stops sampling when xn is still smaller than I_{max}; or (2) it does not stop sampling when xn reaches $I_{max} + 2\epsilon n$. Lemma 6 indicates that the probability that (2) happens is at most $n(\frac{\delta}{2n})^{16}$. We bound the probability that (1) occurs.

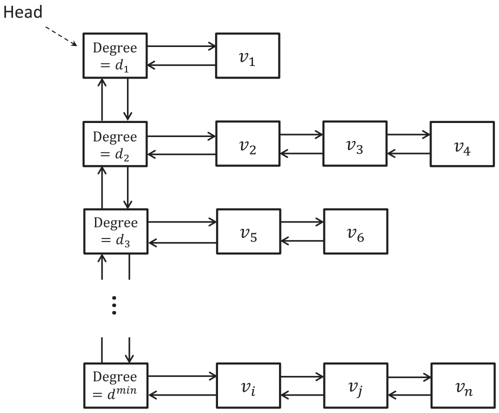


Fig. 3. Linked list structure, where $d_1 > d_2 > d_3 > \cdots > d^{min}$.

Algorithm 2 stops when ${\mathcal{F}_{\mathcal{R}}}^* < x - \epsilon$. According to Lemma 5, if $xn \leq I_{max}$, ${\mathcal{F}_{\mathcal{R}}}^* < x - \epsilon$ happens with probability at most $(\frac{\delta}{2n})^{24}$. Before xn is increased to I_{max}, the test whether ${\mathcal{F}_{\mathcal{R}}}^* \geq x - \epsilon$ is called at most $\frac{48 I_{max} \ln \frac{2n}{\delta}}{n\epsilon^2} = O(\log n)$ times when ϵ and δ are fixed. Thus, the probability that Algorithm 2 stops before xn reaches I_{max} due to ${\mathcal{F}_{\mathcal{R}}}^* < x - \epsilon$ is at most $O(\log n) * (\frac{\delta}{2n})^{24}$.

Putting all things together and applying the Union bound, the failure probability is at most $O(\log n) * (\frac{\delta}{2n})^{24} + n(\frac{\delta}{2n})^{16} = o(\frac{1}{n^{14}})$. □

When the network is updated, the value of I_{max} may change. Thus, after we update the random RR sets, we call Algorithm 2 to ensure that we have enough but not too many random RR sets. In addition, if I_{max} decreases dramatically, which means the current sample size is too large, we abandon some RR sets. Specifically, if ${\mathcal{F}_{\mathcal{R}}}^* < x - \epsilon$, which means with very high probability that $I_{max} < xn$, we keep deleting the last RR set from $\mathcal{R}$ until if deleting the current last RR set leads to ${\mathcal{F}_{\mathcal{R}}}^* > x - \epsilon$ (see Algorithm 3).

Algorithm 3. Deleting Redundant Random RR Sets for Top-K Influential Individuals

```
Input: $G = \langle V, E, w \rangle$, $\epsilon$, $\delta$ and $\mathcal{R}$ which is a set of random RR sets
Output: $\mathcal{R}$
while ${\mathcal{F}_{\mathcal{R}}}^* < x - \epsilon \wedge |\mathcal{R}| > \frac{48 \times 4\epsilon}{\epsilon^2} \ln \frac{2n}{\delta}$ do
    $h \leftarrow$ the last RR set of $\mathcal{R}$
    Delte $h$ from $\mathcal{R}$
    if ${\mathcal{F}_{\mathcal{R}}}^* \geq x - \epsilon \vee |\mathcal{R}| < \frac{48 \times 4\epsilon}{\epsilon^2} \ln \frac{2n}{\delta}$ then
        Add $h$ back to $\mathcal{R}$
        break
    end if
end while
return $\mathcal{R}$
```

Combining Theorem 4 and applying the Union bound, we have that, with probability at least $1 - \delta - o(\frac{1}{n^{14}})$, our maintenance of RR sets ensures that by setting the filtering threshold ${\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{2}$, the set of nodes found includes all real top-K influential nodes and does not have any nodes such that $I_u < I^k - \epsilon n$.

6.3 Improving Precision

According to Theorem 4, we filter out nodes such that $\mathcal{F}_{\mathcal{R}}(u) < {\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{2}$. Using Theorem 3, we can further improve the filtering threshold to make the precision higher.

Theorem 6. *After using Algorithms 2 and 3 to adjust the number of RR sets, with probability at least $1 - 2\delta - o(\frac{1}{n^{14}})$, the following conditions hold.*

1) *If $I_u \geq I^k$, then $\mathcal{F}_{\mathcal{R}}(u) \geq {\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \frac{\epsilon_1}{2}$*
2) *If $I_u < I^k - \epsilon n$, then $\mathcal{F}_{\mathcal{R}}(u) < {\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \frac{\epsilon_1}{2}$*

where $\epsilon_1 = \sqrt{\frac{{\mathcal{F}_{\mathcal{R}}}^{k - \frac{\epsilon}{4}}}{4x}\epsilon} \leq \frac{\epsilon}{2}$.

Proof. According to Theorem 5, after adjusting the number of RR sets by Algorithms 2 and 3, with probability $1 - o(\frac{1}{n^{14}})$, we have $M = \frac{48x}{\epsilon^2} \ln \frac{2n}{\delta} \geq \frac{48 I_{max}}{n\epsilon^2} \ln \frac{\delta}{2n}$. When $M \geq \frac{48 I_{max}}{n\epsilon^2} \ln \frac{\delta}{2n}$, with probability at least $1 - \delta$, $n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{4} \leq I^k \leq n{\mathcal{F}_{\mathcal{R}}}^k + \frac{\epsilon n}{4}$ (Lemmas 3 and 4).

Suppose $M = \frac{48x}{\epsilon^2} \ln \frac{2n}{\delta} \geq \frac{48 I_{max}}{n\epsilon^2} \ln \frac{\delta}{2n}$ and $n{\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon n}{4} \leq I^k \leq n{\mathcal{F}_{\mathcal{R}}}^k + \frac{\epsilon n}{4}$. Let $\epsilon_1 = \sqrt{\frac{{\mathcal{F}_{\mathcal{R}}}^{k - \frac{\epsilon}{4}}}{4x}\epsilon} \leq \frac{\epsilon}{2}$. Applying Theorem 3 and setting the threshold $T = ({\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4})n$, with probability at least $1 - \delta$, we have the follows. (1) if $I_u \geq ({\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4})n$, then $\mathcal{F}_{\mathcal{R}}(u) \geq {\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \frac{\epsilon_1}{2}$; and (2) if $I_u < ({\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \epsilon_1)n$, then $\mathcal{F}_{\mathcal{R}}(u) < {\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \frac{\epsilon_1}{2}$. Clearly $I^k \geq ({\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4})n$ and $I^k - \epsilon n \leq n{\mathcal{F}_{\mathcal{R}}}^k - \frac{3\epsilon n}{4} \leq {\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \epsilon_1)n$.

Applying the Union bound, we have that with probability at least $1 - 2\delta - o(\frac{1}{n^{14}})$, the above conditions hold after executions of Algorithms 2 and 3. □

Theorem 6 shows that we can use a tighter filtering threshold ${\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{4} - \frac{\epsilon_1}{2}$ which is no greater than the original one ${\mathcal{F}_{\mathcal{R}}}^k - \frac{\epsilon}{2}$. Meanwhile, the failure probability is only increased by δ at most.

6.4 Maintaining Nodes Ranking Dynamically

Besides efficiently updating RR sets from where accurate estimations of influence spreads of influential nodes can be obtained, how to maintain the set of influential nodes is also an essential building block of influential nodes mining on dynamic networks. A brute force solution is to perform an $O(n\log n)$ sorting every time after an update but the cost may be unacceptably high in practice. To solve this problem, we adopt the data structure for maximum vertex cover in a hyper graph [8]. This data structure can help us maintain all nodes sorted by their estimated influence spreads, which are proportional to their degrees in $\mathcal{H}$. Clearly, if all nodes are sorted, the set of influential nodes are those ones in the top. Fig. 3 shows the data structure.

We maintain all nodes sorted by their degrees in $\mathcal{H}$ (recall that $\mathcal{D}(u)$, the degree of u in $\mathcal{H}$, means how many RR sets contain u). Nodes with the same degree in $\mathcal{H}$ are grouped together and stored in a doubly linked list like in Fig. 3. Moreover, for those nodes, we create a head node which is the start of the linked list containing all nodes with the same given degree. Apparently, the number of head nodes is the number of distinctive values of $\mathcal{D}(u)$ in $\mathcal{H}$. We also maintain all head nodes sorted in a doubly linked list. For each $u \in V$, we maintain its address in the doubly linked lists structure and the corresponding head node. Note that when a RR set is updated, a new RR set is generated or an existing RR set is deleted, $\mathcal{D}(u)$ changes at most by 1 for each u. Thus, every time when $\mathcal{D}(u)$ is updated (increased or decreased by 1), we only need $O(1)$ time to find the head node of the linked list u should be in (if such a head node does not exist now, we can create it and insert it into the doubly linked list of head nodes in $O(1)$ time) and insert it to the next of the head in $O(1)$ time. If after an update, a head node has no nodes after it, we delete it from the

TABLE 2
The Statistics of the Data Sets

Network	#Nodes	#Edges	Average degree
wiki-Vote	7,115	103,689	14.6
Flixster	99,053	977,738	9.9
soc-Pokec	1,632,803	30,622,564	18.8
flickr-growth	2,302,925	33,140,018	14.4
Twitter	41,652,230	1,468,365,182	35.3

doubly linked list of head nodes in $O(1)$ time. Therefore, in total maintaining the linked list data structure only costs $O(1)$ time when the degree of a node changes due to the update of an RR set. With this data structure, we can always maintain all nodes sorted by their degrees in $\mathcal{H}$. Also, retrieving ${\mathcal{F}_{\mathcal{R}}}^*$, which is needed in the frequently called test whether ${\mathcal{F}_{\mathcal{R}}}^* \leq x - \epsilon$, can be done in $O(1)$ time.

7 Experiments

In this section, we report a series of experiments on 5 real networks to verify our algorithms and our theoretical analysis. The experimental results demonstrate that our algorithms are both effective and efficient.

7.1 Experimental Settings

We ran our experiments on 5 real network data sets that are publicly available online (http://konect.uni-koblenz.de/networks/, http://www.cs.ubc.ca/~welu/and http://konect.uni-koblenz.de). Table 2 shows the statistics of the four data sets.

To simulate dynamic networks, for each data set, we randomly partitioned all edges exclusively into 3 groups: E_1 (85 percent of the edges), E_2 (5 percent of the edges) and E_3 (10 percent of the edges). We used $B = \langle V, E_1 \cup E_2 \rangle$ as the base network. E_2 and E_3 were used to simulate a stream of updates.

For the LT model, for each edge (u, v) in the base network, we set the weight to be 1. For each edge $(u, v) \in E_3$, we generated a weight increase update $(u, v, +, 1)$ (timestamps ignored at this time). For each edge $(u, v) \in E_2$, we generated one weight decrease update $(u, v, -, \Delta)$ and one weight increase update $(u, v, +, \Delta)$ where Δ was picked uniformly at random in $[0, 1]$. We randomly shuffled those updates to form an update stream by adding random time stamps. For each data set, we generated 10 different instances of the base network and update stream, and thus ran the experiments 10 times. Note that for the 10 instances, although the base networks and update streams are different, the final snapshots of them are identical to the data set itself.

For the IC model, we first assigned propagation probabilities of edges in the final snapshot, i.e., the whole graph. We set $w_{uv} = \frac{1}{\text{in-degree}(v)}$, where $\text{in-degree}(v)$ is the number of in-neighbors of v in the whole graph. Then, for each edge (u, v) in the base network, we set w_{uv} to $\frac{1}{\text{in-degree}(v)}$. For each edge $(u, v) \in E_3$, we generated a weight increase update $(u, v, +, \frac{1}{\text{in-degree}(v)})$ (timestamps ignored at this time). For each edge $(u, v) \in E_2$, we generated one weight decrease update $(u, v, -, \Delta \frac{1}{\text{in-degree}(v)})$ and one weight increase update $(u, v, +, \Delta \frac{1}{\text{in-degree}(v)})$ where Δ was picked uniformly at random in $[0, 1]$. We randomly shuffled those updates to form an update stream by adding random time stamps. For each dataset we also generated 10 instances.

For the parameters of tracking nodes of influence at least T, we set $\epsilon = 0.0002$, $\delta = 0.001$, and $T = 0.001 \times n$ for the first four data sets. We set $\epsilon = 0.001$, $\delta = 0.001$, and $T = 0.005 \times n$ for the twitter data set. For the top-K influential individuals tracking task, we set $K = 50$, $\delta = 0.001$, and $\epsilon = 0.0005$ for first four data sets. We set $K = 100$, $\delta = 0.001$, and $\epsilon = 0.0025$ for the twitter data set. The reason we have different parameter settings for the twitter data is that it has more influential nodes than other networks.

All algorithms were implemented in Java and ran on a Linux machine of an Intel Xeon 2.00 GHz CPU and 1 TB main memory.

7.2 Effectiveness

We first assess the effectiveness of our techniques.

7.2.1 Verifying Provable Quality Guarantees

A challenge in evaluating the effectiveness of our algorithms is that the ground truth is hard to obtain. The existing literature of influence maximization [5], [9], [11], [15], [16], [21] always use the influence spread estimated by 20,000 times Monte Carlo (MC) simulations as the ground truth. However, such a method is not suitable for our tasks, because the ranking of nodes really matters here. Even 20,000 times MC simulations may not be able to distinguish nodes with close influence spread. As a result, the ranking of nodes may differ much from the real ranking. Moreover, the effectiveness of our algorithms has theoretical guarantees while 20,000 times MC simulations is essentially a heuristic. It is not reasonable to verify an algorithm with a theoretical guarantee using the results obtained by a heuristic method without any quality guarantees.

In our experiments, we only used wiki-Vote and Flixster to run MC simulations and compare the results to those produced by our algorithms. We used 2,000,000 times MC simulations as the (pseudo) ground truth in the hope we can get more accurate results. According to our experiments, even so many MC simulations may generate slightly different rankings of nodes in two different runs but the difference is acceptably small. We only compare results on the identical final snapshot shared by all instances because running MC simulations on multiple snapshots is unaffordable (10 days on the final snapshots of Flixster).

Table 3 reports the recall of the sets of influential nodes returned by our algorithms and the maximum errors of the false positive nodes in absolute influence value. Ave.±SD represents the average value and the standard deviation of a measurement on 10 instances. Our methods achieved 100 percent recall every time as guaranteed theoretically. Moreover, the real errors in influence were substantially smaller than the maximum error bound provided by our theoretical analysis. One may ask why we do not report the precision here. We argue that precision is indeed not a proper measure for our tasks when 100 percent recall is required. Since we can only estimate influence spreads of nodes via a sampling method due to the exact computation being #P-hard, if two nodes have close influence spreads, say $I_u = 100$ and $I_v = 99$, it is hard for a sampling method to tell the difference between I_u and I_v. Thus, if there are many nodes whose influence spreads are just slightly smaller than the threshold, it is hard to achieve a high precision when ensuring 100 percent recall. Moreover, with a high probability, our method guarantees that influence spreads of false positive nodes are not far away from the real threshold. Such small errors are completely acceptable in many real applications.

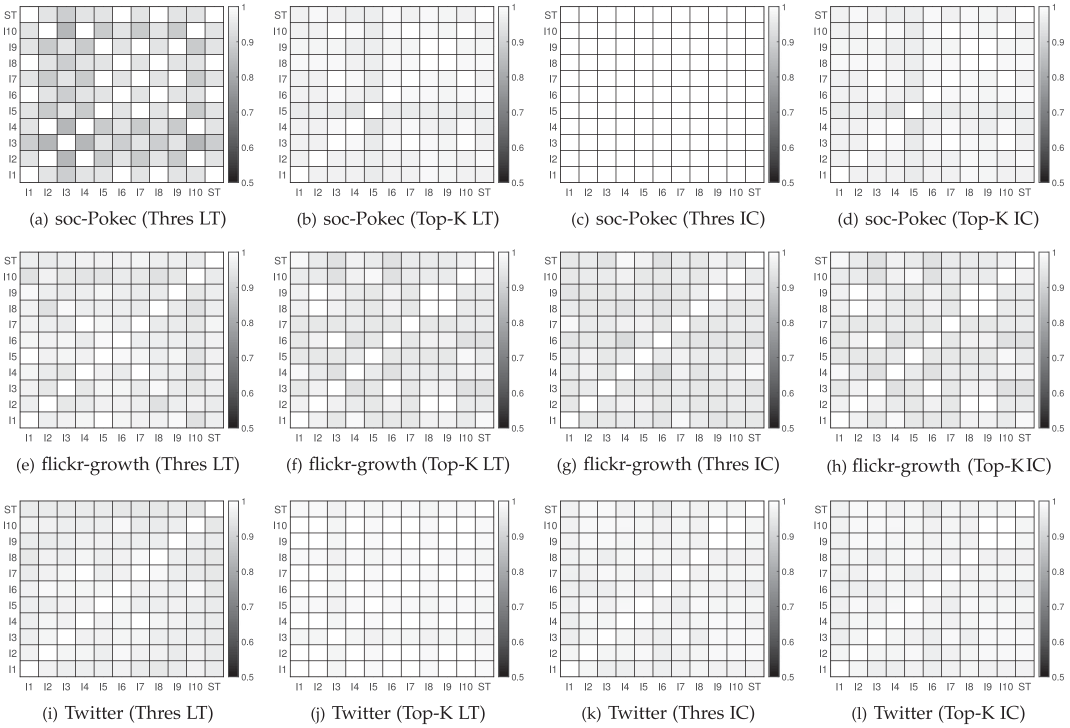
TABLE 3
Recall and Maximum Error

	wiki-Vote			Flixster		
	Theoretical Value (w.h.p.)	Ave. ± SD (LT)	Ave. ± SD (IC)	Theoretical Value (w.h.p.)	Ave. ± SD (LT)	Ave. ± SD (IC)
Recall (Threshold)	100%	100%	100%	100%	100%	100%
Max. Error (Threshold)	$0.0002 * 7115 = 1.423$	0.758 ± 0.033	0.814 ± 0.013	$0.0002 * 99053 = 19.81$	10.81 ± 0.46	11.79 ± 0.85
Recall (Topk-K)	100%	100%	100%	100%	100%	100%
Max. Error (Top-K)	$0.0005 * 7115 = 3.558$	1.254 ± 0.080	1.272 ± 0.090	$0.0005 * 99053 = 49.53$	21.77 ± 0.87	21.17 ± 0.56

The errors are measured in absolute influence value. "w.h.p." is short for "with high probability".

TABLE 4
Estimated Upper Bounds of I_{max} and the Experimental Values

Dataset	LT Model			IC Model		
	xn(Ave. ± SD)	I_{max}	Approx. Ratio(Ave. ± SD)	xn(Ave. ± SD)	I_{max}	Approx. Ratio(Ave. ± SD)
wiki-Vote	57.7297 ± 0.3372	51.770	1.1151 ± 0.0065	55.2079 ± 0.0941	51.7697	1.0664 ± 0.0018
Flixster	456.3168 ± 2.3474	404.2442	1.1288 ± 0.0058	419.9483 ± 1.6652	372.2075	1.1283 ± 0.0045



(a) soc-Pokec (Thres LT) (b) soc-Pokec (Top-K LT) (c) soc-Pokec (Thres IC) (d) soc-Pokec (Top-K IC)

(e) flickr-growth (Thres LT) (f) flickr-growth (Top-K LT) (g) flickr-growth (Thres IC) (h) flickr-growth (Top-KIC)

(i) Twitter (Thres LT) (j) Twitter (Top-K LT) (k) Twitter (Thres IC) (l) Twitter (Top-K IC)

Fig. 4. Similarity among results in different instances.

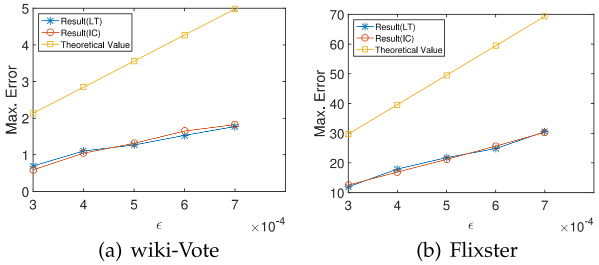
Table 4 reports our estimation of the upper bound of I_{max} on the final snapshot of each network when tracking top-k influential nodes. The results indicate that the upper bound estimated by our algorithm is only a little greater than the real I_{max}.

For the 3 large data sets, we did not run 2,000,000 times MC simulations to obtain the pseudo ground truth since the MC simulations are too costly. Instead, we compare the similarity between the results generated by different instances. Recall that the final snapshots of the 10 instances are the same. If the sets of influential nodes at the final snapshots of the 10 instances are similar, at least our algorithms are stable, that is, insensitive to the order of updates. To measure the similarity between two sets of influential nodes, we adopted the Jaccard similarity.

Fig. 4 shows the results where I1, ... , I10 represent the results of the first, ... , tenth instances, respectively. We also ran the sampling algorithm directly on the final snapshot, that is, we computed the influential nodes directly from the final snapshot using sampling without any updates. The result is denoted by ST. The results show that the outcomes from different instances are very similar, and they are similar to the outcome from ST, too. The minimum similarity in all cases is 87 percent.

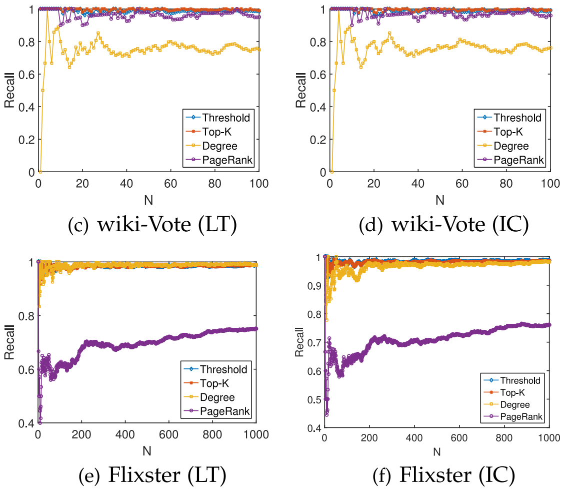
7.2.2 Varying ϵ

For the 2 datasets with (pseudo) ground truth, we also set the error parameter ϵ different values and report results of the Top-K tracking task. Due to limit of space, we omit results of the threshold-based tracking task and results of varying k because they are all similar. In all cases the recall is always 100 percent, we report maximum errors of nodes returned by our algorithm in different settings of ϵ in Fig. 5.



(a) wiki-Vote (b) Flixster

Fig. 5. ϵ versus maximum error.



(c) wiki-Vote (LT) (d) wiki-Vote (IC)

(e) Flixster (LT) (f) Flixster (IC)

Fig. 6. $Recall@N$.

The maximum error is constantly smaller than the theoretical value, and it increases roughly linearly as ϵ increases. Moreover, the theoretical value increases faster than the maximum error.

7.2.3 Comparing with Simple Heuristics

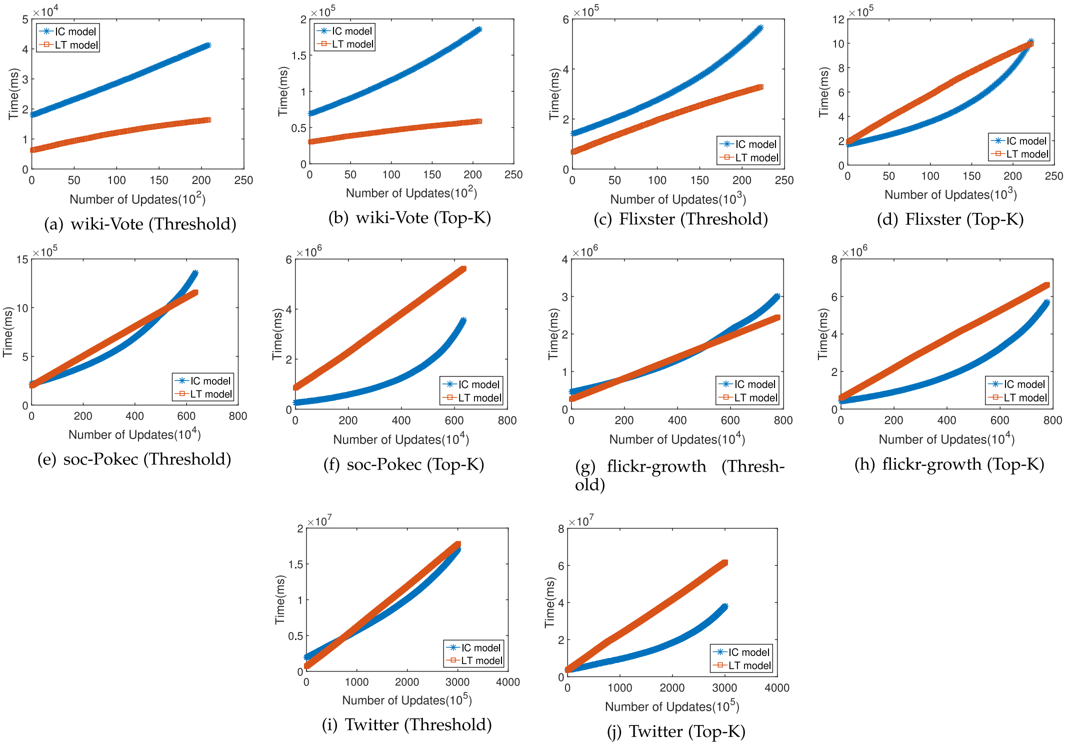
We also compare our algorithms with two simple heuristics, degree and PageRank, which simply return top ranked nodes by degree or PageRank values as influential nodes. The reason we choose these two heuristics is that they both can be efficiently implemented in the setting of dynamic networks. Note that these two heuristics cannot solve the threshold based influential nodes mining problem because they do not know the influence spread of each node.

To compare our algorithms with degree and PageRank heuristics, we report the recall of the top ranked nodes obtained by each method on wiki-Vote and Flixster data sets in Fig. 6. Nodes ranking by 2,000,000 times Monte Carlo simulations is regarded as the (pseudo) ground truth. The measure $Recall@N$ is calculated by $\frac{TP_N}{N}$, where TP_N is the number of nodes ranked top-N by both our algorithms and the ground truth. The results show that the rankings of the top nodes generated by our algorithms constantly have very good quality, while the two heuristics sometimes perform well but sometimes return really poor rankings. Moreover, performance of a heuristic algorithm is not predictable.

7.3 Scalability

7.3.1 Running Time with Respect to Number of Updates

We also tested the scalability of our algorithms. Fig. 7 shows the average running time with respect to the number of updates processed. The average is taken on the running times of the 10 instances. The time spent when the number of updates is 0 reflects the computational cost of running the sampling algorithm on the base network. In Table 5, we also report running time of algorithms on the static final snapshot of each dataset. Clearly, the non-incremental algorithm (rerunning the sampling algorithm from scratch



(a) wiki-Vote (Threshold) (b) wiki-Vote (Top-K) (c) Flixster (Threshold) (d) Flixster (Top-K)

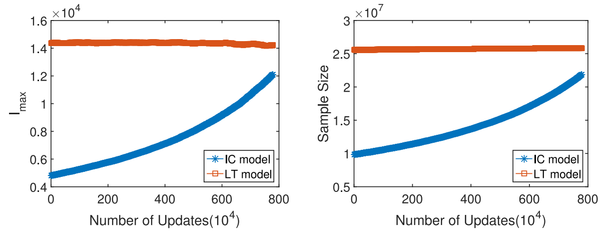
(e) soc-Pokec (Threshold) (f) soc-Pokec (Top-K) (g) flickr-growth (Threshold) (h) flickr-growth (Top-K)

(i) Twitter (Threshold) (j) Twitter (Top-K)

Fig. 7. Scalability.

TABLE 5
Running Time (ms) on Static Networks

Dataset	Threshold		Top-K	
	LT	IC	LT	IC
wiki-Vote	3,635	15,771	17,396	72,447
Flixster	52,970	253,570	116,085	633,498
soc-Pokec	164,330	729,906	751,462	1,790,433
flickr-growth	267,720	1,317,851	1,015,788	3,569,932
Twitter	649,378	5,509,209	3,219,163	19,997,943



(a) I_{max} (b) Sample size M

Fig. 8. I_{max} and M change over time of flickr-growth data.

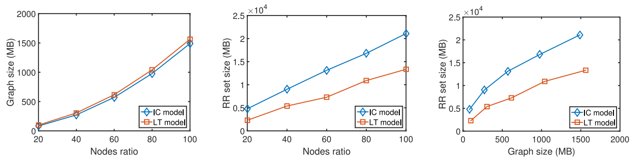
when the network changes) is not competent at all because the running time of processing the base network and the update stream is only several times larger than the running time of processing the whole network, and the number of updates is huge, tens of thousands or even hundreds of millions. This result shows that our incremental algorithm outperforms rerunning the sampling algorithm from scratch by several orders of magnitude.

For the LT model, our algorithm scales up roughly linearly. For the IC model, the running time increases more than linear. This is due to our experimental settings. For the LT model, the sum of propagation probabilities from all in-neighbors of a node is always 1, while in the IC model, at the beginning the sum of propagation probabilities from all in-neighbors is roughly 0.9 but becomes 1 finally. Thus, the spreads of nodes change more dramatically in the IC model than in the LT model. According to our analysis in Section 4.2, the cost of updating the RR sets is proportional to I_v^{t-1}, the influence of v at time $t-1$, and M, the sample size. In the top-k task, M is decided by I_{max}. So the running time curves of the IC model are not linear.

Fig. 8 shows how I_{max} and the sample size M in the top-k task changes over time in the flickr-growth dataset. We do not report results on other datasets because they are all similar.

7.3.2 Memory Usage with Respect to Input Size

We also report the memory usage of our algorithm against the increase of the input graph size. Since the memory needed in Top-K influential nodes mining is usually much higher than the threshold-based mining, we only report results of the Top-K influential nodes mining algorithm. We used the second largest data set, flickr-growth network, to generate some smaller networks. Specifically, we sampled 20, 40, 60 and 80 percent nodes and extract the induced subgraphs. For each sample rate, we sampled 10 subgraphs and for each subgraph we generated a base network and an update stream as we described in Section 7.1. We ran the Top-K influential nodes mining algorithm on those generated data. Fig. 9 reports the average memory storing the input graph and the average peak memory usage of the RR sets against the sample rate. The results show that the size of sampled graph increases super-linearly while the memory of RR sets increases roughly linearly as the sample rate increases. Fig. 9 also shows that the average peak memory used by the RR sets increases sub-linearly as the input graph size increases.



(a) Graph Size (b) RR sets Size (c) RR w.r.t Graph

Fig. 9. Memory usage.

8 Conclusions and Future Work

In this paper, we proposed novel, effective and efficient polling-based algorithms for tracking influential individual nodes in dynamic networks under the Linear Threshold model and the Independent Cascade model. We modeled dynamics in a network as a stream of edge weight updates. We devised an efficient incremental algorithm for updating random RR sets against network changes. For two interesting settings of influential node tracking, namely, tracking nodes with influence above a given threshold and tracking top-k influential nodes, we derived the number of random RR sets we need to approximate the exact set of influential nodes. We reported a series of experiments on 5 real networks and demonstrated the effectiveness and efficiency of our algorithms.

There are a few interesting directions for future work. For example, can we apply similar techniques to other influence models such as the Continuous Time Diffusion Model [7]? Since the Continuous Time Diffusion model has an implicit time constraint, how to efficiently update RR sets according to the time constraint is a critical challenge.

Acknowledgments

Drs. Yang and Pei's research is supported in part by the NSERC Discovery Grant program, the Canada Research Chair program, and the NSERC Strategic Grant program. Drs. Wang and Chen's research is supported in part by the Nature Science Foundation of China (Grant No. U1605251) and the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61325010). All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 57–66.

[2] X. Chen, G. Song, X. He, and K. Xie, "On influential nodes tracking in dynamic social networks," in *Proc. SIAM Int. Conf. Data Mining*, 2015, pp. 613–621.

[3] C. C. Aggarwal, S. Lin, and P. S. Yu, "On influential node discovery in dynamic social networks," in *Proc. SIAM Int. Conf. Data Mining*, 2012, pp. 636–647.

[4] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-I. Kawarabayashi, "Dynamic influence analysis in evolving networks," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1077–1088, 2016.

[5] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 137–146.

[6] W. Chen, L. V. S. Lakshmanan, and C. Castillo, "Information and influence propagation in social networks," *Synthesis Lectures Data Manage.*, vol. 5, no. 4, pp. 1–177, 2013.

[7] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha, "Scalable influence estimation in continuous-time diffusion networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3147–3155.

[8] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2014, pp. 946–957.

[9] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1539–1554.

[10] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *Proc. Int. Conf. Data Mining*, 2010, pp. 88–97.

[11] A. Goyal, W. Lu, and L. V. S. Lakshmanan, "SIMPATH: An efficient algorithm for influence maximization under the linear threshold model," in *Proc. Int. Conf. Data Mining*, 2011, pp. 211–220.

[12] B. Lucier, et al., "Influence at scale: Distributed computation of complex contagion in networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 735–744.

[13] M.-E. G. Rossi, F. D. Malliaros, and M. Vazirgiannis, "Spread it good, spread it fast: Identification of influential nodes in social networks," in *Proc. Int. Conf. World Wide Web*, 2015, pp. 101–102.

[14] Q. Liu, et al., "An influence propagation view of pagerank," *ACM Trans. Knowl. Discovery Data*, vol. 11, no. 3, 2017, Art. no. 30.

[15] E. Cohen, et al., "Sketch-based influence maximization and computation: Scaling up with guarantees," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 629–638.

[16] Y. Tang, et al., "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 75–86.

[17] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "Discovering leaders from community actions," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2008, pp. 499–508.

[18] N. Agarwal, H. Liu, L. Tang, and P. S. Yu, "Identifying the influential bloggers in a community," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2008, pp. 207–218.

[19] J. Weng, E.-P. Lim, J. Jiang, and Q. Hu, "TwitterRank: Finding topic-sensitive influential Twitterers," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2010, pp. 261–270.

[20] M. Cha, et al., "Measuring user influence in Twitter: The million follower fallacy," *Proc. Int. AAAI Conf. Weblogs Social Media*, vol. 10, no. 10–17, 2010, Art. no. 30.

[21] W. Chen, et al., "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 1029–1038.

[22] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 462–470.

[23] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2005, pp. 177–187.

[24] B. Bahmani, et al., "Fast incremental and personalized pagerank," *Proc. VLDB Endowment*, vol. 4, no. 3, pp. 173–184, 2010.

[25] N. Ohsaka, et al., "Efficient pagerank tracking in evolving networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 875–884.

[26] T. Hayashi, et al., "Fully dynamic betweenness centrality maintenance on massive networks," *Proc. VLDB Endowment*, vol. 9, no. 2, pp. 48–59, 2015.

[27] S. Lei, et al., "Online influence maximization," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 645–654.

[28] M. Kimura, et al., "Tractable models for information diffusion in social networks," in *Proc. Eur. Conf. Principles Data Mining Knowl. Discovery*, 2006, pp. 259–271.

[29] A. Goyal, et al., "Learning influence probabilities in social networks," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2010, pp. 241–250.

[30] F. Chung and L. Lu, "Concentration inequalities and martingale inequalities: A survey," *Internet Mathematics*, vol. 3, no. 1, pp. 79–127, 2006.

[31] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *Knowledge-Based Intelligent Information and Engineering Systems.* Berlin, Germany: Springer, 2008, pp. 67–75.

[32] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

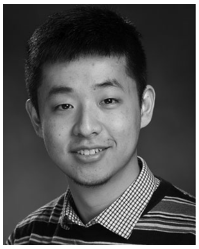
[33] G. Cormode, et al., "Finding frequent items in data streams," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1530–1541, 2008.

[34] M. Riondato and E. Upfal, "Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees," *ACM Trans. Knowl. Discovery Data*, vol. 8, no. 4, 2014, Art. no. 20.

[35] M. Riondato and E. Upfal, "Mining frequent itemsets through progressive sampling with rademacher averages," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1005–1014.

[36] A. Pietracaprina, M. Riondato, E. Upfal, and F. Vandin, "Mining top-K frequent itemsets through progressive sampling," *Data Mining Knowl. Discovery*, vol. 21, no. 2, pp. 310–326, 2010.

[37] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 199–208.



Yu Yang received the BE degree from the Hefei University of Technology, in 2010, and the ME degree from the University of Science and Technology of China, in 2013, both in computer science. He is currently working toward the PhD degree in the School of Computing Science, Simon Fraser University, Canada. His research interests lie in algorithmic aspects of data mining, with an emphasis on managing and mining dynamics of large scale networks.

Zhefeng Wang received the BE degree from the University of Science and Technology of China, China, in 2012. He is currently working toward the PhD degree in the School of Computer Science and Technology, University of Science and Technology of China. His research interests include social network and social media analysis, recommender system, and text mining. He has published several papers in refereed conference proceedings such as SIGIR, KDD, and IJCAI.

Jian Pei is a professor in the School of Computing Science, Simon Fraser University, Canada. His research interests can be summarized as developing effective and efficient data analysis techniques for novel data intensive applications. He is currently interested in various techniques of data mining, Web search, information retrieval, data warehousing, online analytical processing, and database systems, as well as their applications in social networks, health-informatics, business, and bioinformatics. His research has been supported in part by government funding agencies and industry partners. He has published prolifically and served regularly for the leading academic journals and conferences in his fields. He is an associate editor of the *ACM Transactions on Knowledge Discovery from Data*. He is a fellow of the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE). He is the recipient of several prestigious awards.

Enhong Chen received the PhD degree from the University of Science and Technology of China (USTC). He is a professor and vice dean of the School of Computer Science and Technology, USTC. His general area of research includes data mining, personalized recommendation systems, and web information processing. He has published more than 150 papers in refereed conferences and journals. His research is supported by the National Natural Science Foundation of China, National High Technology Research and Development Program 863 of China, etc. He is a program committee member of more than 40 international conferences and workshops. He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE).