Abstract—Influence propagation in networks has enjoyed fruitful applications and has been extensively studied in literature. However, only very limited preliminary studies tackled the challenges in handling highly dynamic changes in real networks. In this paper, we tackle the problem of tracking top-$k$ influential vertices in dynamic networks, where the dynamic changes are modeled as a stream of edge weight updates. Under the popularly adopted linear threshold (LT) model and the independent cascade (IC) model, we address two essential versions of the problem: tracking the top-$k$ influential individuals and finding the best $k$-seed set to maximize the influence spread (Influence Maximization). We adopt the polling-based method and maintain a sample of random RR sets so that we can approximate the influence of vertices with provable quality guarantees. It is known that updating RR sets over dynamic changes of a network can be easily done by a reservoir sampling method, so the key challenge is to efficiently decide how many RR sets are needed to achieve good quality guarantees. We use two simple signals, which both can be accessed in $O(1)$ time, to decide a proper number of RR sets. We prove the effectiveness of our methods. For both tasks the error incurred in our method is only a multiplicative factor to the ground truth. For influence maximization, we also propose an efficient query algorithm for finding the $k$ seeds, which is one order of magnitude faster than the state-of-the-art query algorithm in practice. In addition to the thorough theoretical results, our experimental results on large real networks clearly demonstrate the effectiveness and efficiency of our algorithms.

I. INTRODUCTION

Influence propagation in large networks, such as social networks, has enjoyed fruitful applications. There are two typical kinds of application scenarios. The first type is the well known influence maximization (IM) problem [14], which finds a set of $k$ seed vertices that can collaboratively maximize the influence spread over the whole network. For example, in viral marketing, to promote a product in a social network, a company can use IM to use a small group of influential users to spread the influence in the social network. The second type of application scenarios is to track the top-$k$ most influential nodes in a large network. For example, consider cold-start recommendation in a social network, where 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 by some users in order to obtain hot posts (posts that are widely spread in the social network) at the earliest time. A strategy is to recommend to such a new comer some influential users in the current network. IM cannot find those influential users because IM assumes that 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 needed in this scenario are those who have high individual influence.

More often than not, a network is highly dynamic. For example, in a social network, each vertex is often 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, indeed produce the most significant 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.

Although some aspects of influence propagation in large networks have been extensively investigated, such as IM on static networks [2], [4], [5], [12]–[14], [17], [21], [24], there are only very limited studies on influence computation in dynamic networks [1], [6], most being heuristic. To the best of our knowledge, Ohsaka et al. [20] and we [27] are the first to address influence computation in dynamic networks with provable quality guarantees. However, in [20], how to decide the maintained RR sets are enough to ensure good qualities of seed sets extracted is based on a flawed conclusion of an early version of [2], which now is corrected in the latest version of [2]. Thus, the method in [20] is also heuristic.

We [27] gave solutions to tracking influential individuals with influence greater than a threshold $T$ and tracking top-$k$ influential individuals. Our solutions ensure that, with high probability, the recall is 100% and the smallest influence of the returned set of individuals does not deviate from the true threshold (or the influence of the true $k$-th most influential individual) by an absolute value $\varepsilon n$. A major disadvantage of [27] is that we need to carefully set the parameters to obtain meaningful results. If the threshold $T$ is too high, [27] may return nothing. If the parameter $\varepsilon$, which controls the tolerable absolute error in the top-$k$ task, is not set properly, $\varepsilon n$ may be even greater than the influence of the $k$-th most influential individual, and [27] may return all vertices, which may be meaningless. Thus, to run algorithms in [27] properly, one needs some prior knowledge about the data, which is often not easy to get.

To tackle the obstinate fundamental challenges in influence computation in dynamic networks, in this study, we systematically tackle the two essential tasks of tracking top-$k$ influential vertices: (1) tracking the top-$k$ most influential vertices; and (2) supporting efficient influence maximization queries. For both tasks, our goal is to control the error incurred in our algorithms a multiplicative factor to the ground truth, so that even without prior knowledge about the data, we can still obtain meaningful results by setting a relative error threshold.
Similar to [20] and [27], in this paper, we also maintain a sample of RR sets to approximate influence spread of vertices. Both [20] and [27] point out that updating existing RR sets against an update of the network can be easily done by a reservoir sampling method, and after updating, a critical step is to decide if the current sample size (number of RR sets) is proper to achieve good quality guarantees. Due to the high-speed updates in real networks, such decision has to be made efficiently. Meanwhile the decision should help us maintain as few RR sets as possible.

In this paper, we make several technical contributions. Our foremost contributions are at the theoretical side. We adopt two simple signals, which both can be accessed in $O(1)$ time, to decide a proper sample size for the two tasks. For the individual tracking task, our sample size is very close to the minimum sample size for estimating the $k$ largest individual influence with a small relative error. For influence maximization, our sample size is not big when the seed set size $k$ of IM queries is small but becomes too large to use when $k$ is large. Thus, we also give a practical solution that normally is more efficient than the method in [20], and also effective in practice. In addition to the thorough theoretical results, our experimental results on real networks clearly demonstrate the effectiveness and efficiency of our algorithms. Our methods are more efficient than the state-of-the-art methods as the baselines. The largest network used in the experiments has 41.7 million vertices, almost 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 II. In Section III, we recall the Linear Threshold model and the Independent Cascade model, review the polling-based method for computing influence spread on dynamic networks, and introduce two major tools from probability theory that we will use in this paper. In Section IV, we tackle the problem of tracking top-$k$ influential individuals with limited relative error rates. In Section V, we analyze how to maintain a number of RR sets to support influence maximization queries with good quality guarantees. We also devise an efficient implementation of the greedy algorithm for IM queries. We report the experimental results in Section VI and conclude the paper in Section VII.

II. RELATED WORK

In this section, we briefly review the state of the art about influence computation on both static and dynamic networks.

A. Influence Computation in Static Networks

One major difficulty in influence computation is that computing influence spread is $\#P$-hard under both the LT and IC models [3], [4]. Besides some heuristic methods for estimating influence spread [4], [8], [13], recently, a polling-based method [2], [23], [24] was proposed for influence maximization (IM) under general triggering models. The key idea is to use some “Reverse Reachable” (RR) sets [23], [24] to approximate the true influence spread of vertices. The error of approximation can be bounded with a high probability if the number of RR sets is large enough. Nguyen et al. [19] exploits the Stopping Rule [10] for sampling to further improve the efficiency of the polling based IM algorithms.

B. Heuristics for Tracking Influential Vertices

For evolving networks [15], rather than re-computing from scratch, incremental algorithms are more desirable in analytic tasks. Some studies propose heuristic methods for influence computation on dynamic networks. Aggarwal et al. [1] explored how to find a set of vertices that has the highest influence within a time window $[t_0, t_0 + h]$ by modeling influence propagation as a non-linear system. Chen et al. [6] investigated incrementally updating a seed set for influence maximization under the Independent Cascade model. Both [1], [6] cannot deal with real-time updates of dynamic networks, since in [1] the snapshot of the network in the time window $[t_0, t_0 + h]$ needs to be extracted before the mining process, and [6] has the time complexity $O(m)$ to deal with one update of the network, where $m$ is the number of edges.

Recently, Ohsaka et al. [20] studied maintaining a collection of RR sets $\mathcal{R}$ over a stream of network updates under the IC model such that $(1 - 1/e - \epsilon)$-approximation IM queries can be achieved with probability at least $1 - \frac{1}{e}$. Specifically, [20] maintains the invariant $C(\mathcal{R}) = \Theta((m + n) \log n)$, where $C(\mathcal{R})$ is the number of edges traveled for generating all RR sets in $\mathcal{R}$. Unfortunately, setting $C(\mathcal{R}) = \Theta((m + n) \log n)$ is based on a flawed conclusion in an early version of [2] and the correct one is to set $C(\mathcal{R}) = \Theta(k \frac{(m + n) \log n}{\epsilon^2})$. Thus, [20] does not have provable quality guarantees. Moreover, even $\Theta(m/n \log n)$ is a huge number so in the experiments of [20], $C(\mathcal{R})$ is empirically set to $32(m + n) \log n$, which is far less than $\Theta((m + n) \log n)$ but still a very large number.

C. Tracking Influential Vertices with Quality Guarantees

Our previous work [27] is the first to track influential vertices in dynamic networks with quality guarantees. We [27] tackled two versions of tracking influential vertices, namely tracking vertices with influence greater than a threshold $T$ and tracking the top-$k$ influential vertices. With high probability, [27] achieves 100% recall and a bounded error of all false positive vertices, where the error of a false positive vertex $u$ is the threshold (the $k$-th largest influence spread in the top-$k$ task) minus the influence of $u$. The major disadvantage of [27] is that the parameters have to be carefully set, otherwise the returned result may be overwhelming and undesirable. More specifically, for the threshold-based task, if the threshold $T$ is set to much larger than the greatest individual influence, the algorithm may return nothing. For the top-$k$ task, since the error in [27] is an absolute error $\epsilon n$, and we do not know the value of $I^k$ beforehand, where $I^k$ is the $k$-th greatest individual influence, one may set $\epsilon n$ to even greater than $I^k$ and then [27] may return all vertices.

Recently, Wang et al. [26] proposed to maintain an $\epsilon(1-\beta)$-approximation solution of influence maximization dynamically. However, rather than a propagation model like the IC or the LT model, [26] employs the simple reachability in a social action graph as influence. Moreover, the input of [26] is a stream of social actions which is different from edge propagation probability updates in [20], [27] and this paper. Thus, [26] cannot be used to track influential vertices under propagation models like the IC or LT model.
The optimal degree of \( \Theta \) in influence spread of the triggering models \([14]\) like the LT 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 \(I(S)\) is the expected number of vertices reachable from \(S\) via live edges.

### B. Polling Estimation of Influence Spread

Computing influence spread is \#P-hard under both the LT model and the IC model \([3, 4]\). Recently, a polling-based method \([2, 23, 24]\) was proposed for approximating influence spread of triggering models \([14]\) like the LT model and the IC model. Here we briefly review the polling method for computing influence spread.

Given a social network \(G = (V,E,w)\), a poll is conducted as follows: we pick a vertex \(v \in V\) at random and then try to find out which vertices are likely to influence \(v\). We run a Monte Carlo simulation of the equivalent “live-edge” process. The vertices 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 (for Reverse Reachable) set.

Let \(\mathcal{R} = \{R_1, R_2, ..., R_M\}\) be a set of random RR sets generated by \(M\) polls. For a set of vertices \(S\), denote by \(\mathcal{D}(S)\) the degree of \(S\) in \(\mathcal{R}\), which is the number of RR sets that contain at least one vertex in \(S\). By the linearity of expectation, \(\mathcal{D}(S)\) is an unbiased estimator of \(I(S)\) \([2]\). Thus, in the polling method, \(\frac{\mathcal{D}(S)}{M}\) is used to approximate \(I(S)\). How to decide the sample size \(M\) often is the key point for efficient computation in tasks related to influence.

### C. Updating RR Sets on Dynamic Networks

We \([27]\) modeled the updates of an influence network as a stream of edge weight updates, where an edge weight update is depicted as a 5-tuple \((u,v,+,-,\Delta,t)\). \((u,v)\) denotes the edge to be updated, \(+\) and \(-\), respectively, indicate whether \(w_{uv}\) is increased or decreased, \(\Delta > 0\) is the amount of change and \(t\) is the time stamp. \([27]\) pointed out that a RR set under the LT model is a random path, and both \([20]\) and \([27]\) illustrated that a RR set in the IC model is a random connected component. When an edge weight update \((u,v,+,-,\Delta,t)\) comes, Algorithm 1 gives a framework that updates RR sets correspondingly so that \(\frac{\mathcal{D}(S)}{M}\) remains an unbiased estimator of \(I(S)\) for any \(S \subseteq V\).

Line 1 can be easily achieved by maintaining an inverted index on all RR sets so that we can access all RR sets passing a specific vertex. Line 2 can also be easily tackled by a method similar to Reservoir Sampling \([25]\), for both LT and IC models.

### III. PRELIMINARIES

In this section, we review the two most widely used influence models and the polling method for estimating influence spread. We also briefly illustrate how to update RR sets in the polling method over a stream of edge weight updates. The two major probabilistic methods used to derive sample size in this paper are introduced at the end of this section. For readers’ convenience, Table I lists some frequently used notations.

#### A. The LT and IC Models

Consider a directed social network \(G = (V,E,w)\), 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 \(w_{uv} \in [0, +\infty)\).

1) The LT Model: In the Linear Threshold (LT) model \([14]\), each vertex \(v \in V\) also carries a weight \(w_v\), which is called the self-weight of \(v\). Denote by \(N^0(v)\) the set of \(v\)'s in-neighbors. The influence probability of an edge \((u,v)\) is \(p_{uv} = \frac{w_v}{w_u}\). Clearly, for \(v \in V\), \(\sum_{u \in N^0(v)} p_{uv} \leq 1\).

In the LT model, given a seed set \(S \subseteq V\), the influence is propagated in \(G\) as follows. First, every vertex \(u\) randomly selects a threshold \(A_u \in [0,1]\), which reflects our lack of knowledge about users’ true thresholds. Then, influence propagates iteratively. Denote by \(S_t\) the set of vertices that are active in step \(t\) \((i = 0, 1, \ldots)\). Set \(S_0 = S\). In each step \(i \geq 1\), an inactive vertex \(v\) becomes active if \(\sum_{u \in N^0(v)} p_{uv} \geq A_v\). The propagation process terminates in step \(t\) if \(S_t = S_{t-1}\).

Let \(I(S)\) be the expected number of vertices 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 vertex \(u\). Kempe et al. \([14]\) proved that the LT model is equivalent to a “live-edge” process where each vertex \(v\) picks at most one incoming edge \((u,v)\) with probability \(p_{uv}\). Consequently, \(v\) does not pick any incoming edges with probability \(1 - \sum_{u \in N^0(v)} p_{uv} = \frac{w_v}{w_u}\). All edges picked are “live” and the others are “dead”. Then \(I(S)\) is the expected number of vertices reachable from \(S \subseteq V\) through live edges.

2) The IC Model: In the Independent Cascade (IC) model \([14]\), for an edge \((u,v)\), \(w_{uv}\) is the influence probability \((0 \leq w_{uv} \leq 1)\). Given a seed set \(S\), the influence propagates in \(G\) in a way different from the LT model. Denote by \(S_t\) the set of vertices that are active in step \(i\) \((i = 0, 1, \ldots)\). Set \(S_0 = S\). In each step \(i \geq 1\), each vertex \(u\) that is newly activated in step \(i - 1\) has a single chance to influence its inactive neighbor \(v\) with an independent probability \(w_{uv}\). The propagation process terminates in step \(t\) when \(S_t = S_{t-1}\).

Similar to the LT model, the influence spread \(I(S)\) is the expected number of vertices that are finally active when the seed set is \(S\). The equivalent “live-edge” process \([14]\) 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 \(I(S)\) is the expected number of vertices reachable from \(S\) via live edges.

#### TABLE I. FREQUENTLY USED NOTATIONS.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_u)</td>
<td>The influence spread of a single vertex (u)</td>
</tr>
<tr>
<td>(S)</td>
<td>A set of (M) random RR sets</td>
</tr>
<tr>
<td>(D(u))</td>
<td>The degree of (u \in V) in (G), or equivalently the number of (RR) sets containing (u)</td>
</tr>
<tr>
<td>(D(S))</td>
<td>The degree of (S \in V), the number of (RR) sets containing at least one vertex from the set (S)</td>
</tr>
<tr>
<td>(I^k)</td>
<td>Influence spread of the (k)-th most influential individual vertex</td>
</tr>
<tr>
<td>(S_{\text{opt}})</td>
<td>The (k)-seed set produced by running the greedy algorithm (Algorithm 3) on (RR) sets</td>
</tr>
<tr>
<td>(S_{\text{opt}}^k)</td>
<td>The optimal (k)-seed set, (S_{\text{opt}}^k = \text{argmax}_{S \subseteq V,</td>
</tr>
<tr>
<td>(Y(\epsilon, \delta))</td>
<td>(Y(\epsilon, \delta) = \frac{4(1-2\epsilon)}{\delta^2})</td>
</tr>
<tr>
<td>(Y_1(\epsilon, \delta))</td>
<td>(Y_1(\epsilon, \delta) = 1 + (1+\epsilon)Y(\epsilon, \delta))</td>
</tr>
</tbody>
</table>
The key point of Algorithm 1 is Lines 3 and 6, that is, how to decide the proper current sample size $M$. $M$ varies in different tasks of influence computation. Also, [27] analyzed that the cost of dealing with an edge weight update is proportional to the sample size $M$ for both the LT and IC model. Thus, we remark that efficiently deciding a small and proper sample size $M$ is the core of influence computation tasks on dynamic networks.

**D. Martingale Inequalities and Stopping Rule Theorem**

In this section, we introduce two major tools from probability theory that help us derive the proper sample size $M$.

Let $R_1, R_2, ..., R_M$ be a sequence of random RR sets generated by $M$ polls, where $M$ can also be a random variable. Tang et al. [24] proved that the corresponding sequence $Z_1 = \sum_{i=1}^{1} (x_i - \frac{I(S)}{n}), Z_2 = \sum_{i=1}^{2} (x_i - \frac{I(S)}{n}), ..., Z_M = \sum_{i=1}^{M} (x_i - \frac{I(S)}{n})$ is a martingale [7], where $x_i = 1$ if $S \cap R_i \neq \emptyset$ and $x_i = 0$ otherwise. We have $E[\sum_{i=1}^{M} x_i] = E[D(S)] = \frac{M(I(S))}{n}$. The following results [24] show how $E[\sum_{i=1}^{M} x_i]$ is concentrated around $\frac{M(I(S))}{n}$, when variables $Z_1, Z_2, ..., Z_M$ may be weakly dependent due to the stopping condition on $M$.

**Corollary 1** ([24]). For any $\delta > 0$,

\[
\Pr \left[ \sum_{i=1}^{M} x_i - M p \geq \xi M p \right] \leq \exp \left( -\frac{\xi^2}{2 + 4 \xi} M p \right)
\]

\[
\Pr \left[ \sum_{i=1}^{M} x_i - M p \leq -\xi M p \right] \leq \exp \left( -\frac{\xi^2}{2} M p \right)
\]

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

Besides the above martingale inequalities, we also introduce a Stop-Rule Sampling method for obtaining an $(\epsilon, \delta)$ estimation$^1$ of the expectation of a Bernoulli random variable.

Define $Y(\epsilon, \delta) = \frac{4(1-\epsilon)\ln \frac{2}{\delta}}{\epsilon^2}$ and $Y_1(\epsilon, \delta) = 1 + (1+\epsilon)Y(\epsilon, \delta)$. Dagum et al. [10] proposed a Stopping Rule algorithm (Algorithm 2) to obtain an $(\epsilon, \delta)$ estimation of the mean of a Bernoulli random variable.

**Corollary 2** (Stopping Rule Theorem [10]). Let $Z$ be a Bernoulli random variable with $\mu_Z = E[Z] > 0$. Let $\bar{\mu}_Z$ be the estimate produced and let $M$ be the number of experiments that the Stopping Rule algorithm runs with respect to $Z$ on the input $\epsilon$ and $\delta$. Then, (1) $\Pr \left[ \bar{\mu}_Z \geq (1-\epsilon)\mu_Z \right] \geq 1 - \delta$ and $\Pr \left[ \bar{\mu}_Z \leq (1+\epsilon)\mu_Z \right] \geq 1 - \delta$. (2) $E[M] \leq Y_1(\epsilon, \delta)/\mu_Z$.

Corollary 2 implies that with probability at least $1 - \delta$, $Y_1(\epsilon, \delta)/\mu_Z \leq M \leq Y_1(\epsilon, \delta)/(1+\epsilon)\mu_Z$. It is obvious that $Y(\epsilon, \delta)$ is monotonically decreasing with respect to $\epsilon$, thus $Y_1(\epsilon, \delta)/M$ is a slightly better estimate of $\mu_Z$ because $Y_1(\epsilon, \delta) = [1 + (1+\epsilon)Y(\epsilon, \delta)]$, where $\delta \leq \epsilon$.

Note that in the Algorithm 2, in the last experiment the random variable $Z_M$ must be 1. Thus, the stopping time of Algorithm 2 is the first time when the number of positive samples equals $Y_1(\epsilon, \delta)$. We prove that by relaxing this condition a little bit, that is, we only need a sufficient number of positive samples but the last sample does not have to be 1, we still can bound the error in the estimation tightly.

**Theorem 1.** Let $Z$ be a Bernoulli random variable with $\mu_Z = E[Z] > 0$. Let $Z_1, Z_2, ..., Z_M$ be independently and identically distributed according to $Z$. Let $M$ be a stopping time ($M$ is a random variable) for this sequence. Given $(\epsilon, \delta)$, if $\sum_{i=1}^{M} Z_i = Y_1(\epsilon, \delta)$, then $Pr \left[ \frac{\sum_{i=1}^{M} Z_i}{M} \leq (1+\epsilon)\mu_Z \right] \geq 1 - \frac{\delta}{2}$ and $Pr \left[ \frac{\sum_{i=1}^{M} Z_i}{M} \leq (1-\epsilon)\mu_Z \right] \geq 1 - \frac{\delta}{2}$.

**Proof:** Suppose $M_1$ is the stopping time when the first time $\sum_{i=1}^{M_1} Z_i = Y_1(\epsilon, \delta)$ and $M_2$ is the stopping time when the first time $\sum_{i=1}^{M_2} Z_i = Y_1(\epsilon, \delta) + 1$. Clearly, $M_1 \leq M \leq M_2$. According to the Stopping Rule theorem, we have $Pr \left[ M \geq Y_1(\epsilon, \delta)/\mu_Z \right] \geq 1 - \delta/2$ and $Pr \left[ M \leq Y_1(\epsilon, \delta) + 1/\mu_Z \right] \geq 1 - \delta/2$. Since $\sum_{i=1}^{M} Z_i = Y_1(\epsilon, \delta)$, we have $Pr \left[ \frac{\sum_{i=1}^{M} Z_i}{M} \leq (1+\epsilon)\mu_Z \right] \geq 1 - \frac{\delta}{2}$ and $Pr \left[ \frac{\sum_{i=1}^{M} Z_i}{M} \geq (1-\epsilon)\mu_Z \right] \geq 1 - \frac{\delta}{2}$.

As illustrated, the random variable $x_i$ ($x_i = 1$ if $S \cap R_i \neq \emptyset$ and $x_i = 0$ otherwise) is clearly a Bernoulli random variable with mean $\frac{I(S)}{n}$. Thus, we can use Theorem 1 to analyze the quality of estimations of influence spreads based on the maintained RR sets.

**IV. TRACKING TOP-k INDIVIDUALS**

**A. Algorithm**

First of all, tracking influential individuals is not an instance of the Heavy Hitters problem [9]. Detailed reasons are illustrated in [27].

Due to the #P-hardness of computing influence spread [3], [4], it is unlikely that we can find in polynomial time the exact set of top-k influential individual vertices. Thus, we turn to algorithms that allow controllable small errors. Specifically, we ensure that the recall of the set of vertices found by our algorithm is 100% and we tolerate some false positive vertices. Moreover, the influence spreads of those false positive vertices should take a high probability to have a lower bound that is not much smaller than $I^k$, the influence spread of the $k$-th most influential vertex. We call $max_{\delta \leq 5}(I^k - I_{\delta})$ the error of $S$, where $S$ is returned by a top-k influential vertices finding algorithm.

We give a simple solution to track top-k individuals by a collection of RR sets $\mathcal{R}$. We split $\mathcal{R}$ into two disjoint parts $\mathcal{R}_1$ and $\mathcal{R}_2$. $\mathcal{R}_1$ is for deriving an upper bound and a lower bound of $I^k$, and a proper sample size of $\mathcal{R}_2$. We use $\mathcal{R}_2$ and the lower bound of $I^k$ to return a set of influential individuals. Denote by $D_1(u)$ and $D_2(u)$ the degrees of $u$ (number of RR sets containing $u$) in $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. Let $M_1 = |\mathcal{R}_1|$ and $M_2 = |\mathcal{R}_2|$. Our algorithm works as follows,

1) Update the RR sets in $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ when the network structure changes using the method in [27].
2) Maintain the invariant $D_k^1 = [\Upsilon_1(e, \frac{\delta}{n})]$, where $D_k^1$ is the $k$-th largest degree of the vertices in $R_1$.
3) Also maintain the invariant $M_k = M_1$.
4) When a top-$k$ individuals query is issued, return all vertices $u$ such that $D_2(u) \geq T = \frac{1}{1+\epsilon} D_k^1$.

In our method, $D_k^1$ is the signal used to decide if the current sample size is appropriate. When $D_k^1$ is not $[\Upsilon_1(e, \frac{\delta}{n})]$, we adjust the sizes of $R_1$ and $R_2$ by adding or deleting some RR sets. In Section IV-C, we propose an efficient way to retrieve $D_k^1$ in $O(1)$ time such that whether the current sample size is proper can be efficiently decided.

B. Analysis of Sample Size

We show the effectiveness of our algorithm that it can achieve the goals of 100% recall and an $O(\epsilon)$ relative error with high probability. We also demonstrate that our algorithm is efficient in sample size.

The analysis of the effectiveness of our algorithm consists of two major steps. The first step is to bound $I^k$ using $D_k^1$. Once we can bound $I^k$ within a small range, we can set a safe threshold $T$ on $\frac{n D_1(v)}{M_1}$ to find the top-$k$ influential individuals and filter out the vertices that very likely are not ranked within top-$k$. As introduced above, we set the threshold $T = \frac{1}{1+\epsilon} D_k^1$. The second step is to use the upper bound and the lower bound of $I^k$ to derive the error rate of the false positive vertices, that is, how much the influence of a false positive vertex may be smaller than $I^k$.

First, we prove that for every $v \in V$, $\frac{n D_1(v)}{M_1}$ can be used to bound its influence $I_v$ with high probability. For each vertex $v$, we can calculate $\varepsilon_v$ such that $D_1(v) = Y_1(e_v, \frac{\delta}{n})$. We split the vertices in $V$ into two parts, $V_1$ and $V_2$, where $V_1 = \{v \mid \varepsilon_v \leq 0.6\}$ and $V_2 = \{v \mid \varepsilon_v > 0.6\}$. We prove that with high probability for every $v \in V_1$, its influence $I_v$ has an upper bound and a lower bound, and for every $v \in V_2$, $I_v$ has an upper bound.

**Lemma 1.** Suppose in $R_1$, $D_k^1 = [\Upsilon_1(e, \frac{\delta}{n})]$. For any vertex $v \in V$, we calculate a corresponding $\varepsilon_v$ such that $D_1(v) = Y_1(e_v, \frac{\delta}{n})$. When $\varepsilon \leq \frac{1}{4}$ and $\delta \leq \frac{1}{2}$, if $\varepsilon_v \leq 0.6$, $Pr[I_v \geq \frac{n D_1(v)}{(1+\epsilon)M_1}] \geq 1 - \frac{6}{2n}$ and $Pr[I_v \leq \frac{n D_1(v)+1}{(1+\epsilon)M_1}] \geq 1 - \frac{\delta}{2n}$. (2) if $\varepsilon_v > 0.6$, $Pr[I_v < \frac{n D_1(n)}{M_1}] \geq 1 - \frac{\delta}{2n}$.

**Proof:** The first part can be directly obtained by applying Theorem 1. We prove the second part. Note that $Y_1(e, \delta)$ is decreasing with respect to $e$. When $\varepsilon \leq \frac{1}{4}$, for every vertex such that $\varepsilon_v > 0.6$, we have $D_1(v) < Y_1(0.6, \frac{\delta}{n}) < \frac{1}{2} Y_1(e_v, \frac{\delta}{n})$. We prove that, with high probability, for every $v$, $D_1(v) < \frac{1}{2} Y_1(e_v, \frac{\delta}{n})$ implies $I_v < \frac{n Y_1(e_v, \frac{\delta}{n})}{M_1}$.

If $I_v \geq \frac{n Y_1(e_v, \frac{\delta}{n})}{M_1}$, according Corollary 1, we have

$$Pr[D_1(v) < \frac{1}{2} Y_1(e_v, \frac{\delta}{n})] \leq Pr[D_1(v) \leq \frac{1}{2} I_v \leq \frac{2}{n}]$$

$$\leq \exp\left\{-\frac{1}{8} M_1 Y_1(e_v, \frac{\delta}{n})\right\}$$

$$\leq \exp\left\{-\frac{(e-2) \ln 2}{2e^2} \frac{\delta}{n}\right\}$$

Since $\varepsilon \leq \frac{1}{4}$, we have that if $I_v \geq \frac{n Y_1(e_v, \frac{\delta}{n})}{M_1}$, $Pr[D_1(v) < \frac{1}{2} Y_1(e_v, \frac{\delta}{n})] \leq \frac{(\frac{\delta}{n})^\delta}{2^\delta} \leq \frac{\delta}{2n}$. This means if $D_1(v) < \frac{1}{2} Y_1(e_v, \frac{\delta}{n})$, then with probability at least $1 - \frac{\delta}{2n}$, we have $I_v < \frac{n Y_1(e_v, \frac{\delta}{n})}{M_1}$.

We now derive a lower bound of $I^k$ when $D_k^1 = [\Upsilon_1(e, \frac{\delta}{n})]$ in our maintained $R_1$. The intuition is if we can find at least $k$ vertices whose influence spreads probably are no smaller than $T$, then $T$ is a lower bound of $I^k$ with high probability.

**Lemma 2.** If $\varepsilon \leq \frac{1}{4}$, $\delta \leq \frac{1}{2}$ and $D_k^1 = [\Upsilon_1(e, \frac{\delta}{n})]$, then $Pr[I^k \geq \frac{n D_k^1}{(1+\epsilon)M_1}] \geq 1 - \frac{\delta}{2}$.

**Proof:** According to Lemma 1, and applying the union bound, we have that with probability at least $1 - \frac{\delta}{2}$, for all $v$ such that $D_1(v) = Y_1(e_v, \frac{\delta}{n})$ and $\varepsilon_v \leq 0.6$, $\frac{n D_1(v)}{M_1} \leq (1 + \varepsilon_v) I_v$ so $I_v \geq \frac{n D_1(v)}{(1+\epsilon)M_1}$. Clearly there are at least $k$ vertices such that $\frac{n D_1(v)}{M_1} \geq \frac{n D_k^1}{(1+\epsilon)M_1}$. Also, $\varepsilon_v$ is decreasing with respect to $D_1(v)$. Thus, if $D_1(v) \geq D_k^1$, then $\frac{n D_1(v)}{(1+\epsilon)M_1} \geq \frac{n D_k^1}{(1+\epsilon)M_1}$. Therefore, with probability at least $1 - \frac{\delta}{2}$, there are at least $k$ vertices whose influence spreads are no smaller than $\frac{n D_k^1}{(1+\epsilon)M_1}$. So we get that

$$Pr[I^k \geq \frac{n D_k^1}{(1+\epsilon)M_1}] \geq 1 - \frac{\delta}{2}.$$
Note that \( \Pr\{k^T \geq \frac{nD^1_T}{(1+\epsilon)M_1} \} \geq 1 - \frac{\delta}{2} \) implies \( \Pr\{M_1 \geq \frac{n[Y_1(\epsilon, \frac{\delta}{2})]}{(1-\epsilon)^k} \} \geq 1 - \frac{\delta}{2} \). Recall that in our algorithm, we make \( |R_2| = M_2 = M_1 \). Thus, \( \Pr\{M_2 \geq \frac{n[Y_1(\epsilon, \frac{\delta}{2})]}{(1+\epsilon)^k} \} \geq 1 - \frac{\delta}{2} \). We show that when \( M_2 \geq \frac{n[Y_1(\epsilon, \frac{\delta}{2})]}{(1+\epsilon)^k} \), for every true top-k vertex \( u \), it is very likely that \( \frac{nD^2_2(u)}{M_2} \) is not much smaller than \( k^T \). We also show that, for every vertex \( v \) such that \( I_v \) is sufficiently smaller than \( k^T \), its estimation \( \frac{nD^2_2(u)}{M_2} \) does not deviate from the expectation \( I_v \) by \( \epsilon k^T \).

**Lemma 4.** For \( M_2 \geq \frac{n[Y_1(\epsilon, \frac{\delta}{2})]}{(1+\epsilon)^k} \) RR sets in \( R_2 \), with probability at least \( 1 - \frac{\delta}{2} \), we have (1) for every \( v \) such that \( I_v \geq k^T \), \( \frac{nD^2_2(v)}{M_2} \geq (1-\epsilon)k^T \); and (2) for every \( v \) such that \( I_v \leq (1-2\epsilon)k^T \), \( \frac{nD^2_2(v)}{M_2} \leq I_v + \epsilon k^T \).

**Proof:** For each \( v \in V \),

1) If \( I_v \geq k^T \), we have
\[
\Pr\left( \frac{nD^2_2(v)}{M_2} \leq (1-\epsilon)k^T \right) \leq \Pr\left( \frac{nD^2_2(v)}{M_2} \leq (1-\epsilon)I_v \right) \\
\leq \exp\left( -\frac{\epsilon^2 I_v}{2M_2} \right) \leq \frac{\delta}{2n}
\]

2) If \( I_v \leq (1-2\epsilon)k^T \), we have
\[
\Pr\left( \frac{nD^2_2(v)}{M_2} \geq I_v + \epsilon k^T \right) \leq \exp\left( -\frac{(\epsilon k^T)^2}{2M_2} \right) \\
\leq \exp\left( -\frac{\epsilon^2 k^T I_v}{2M_2} \right) \leq \frac{\delta}{2n}
\]

Apply the Union bound, the theorem holds.

Based on Corollary 3 and Lemma 4, we have the following theorem that shows the effectiveness of our algorithm.

**Theorem 2.** When \( \epsilon \leq \frac{1}{3} \) and \( \delta \leq \frac{1}{4} \), our algorithm described in Section IV-A returns a set of vertices \( S \) such that all real top-k vertices are included, and \( \min_{u \in S} I_u \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \), where \( B = 4(\epsilon - 2) \ln \frac{n}{\delta} \).

**Proof:** Based on Corollary 3 and Lemma 4, when \( \epsilon \leq \frac{1}{3} \), \( \delta \leq \frac{1}{4} \), \( D^1_T = \left[ Y_1(\epsilon, \frac{\delta}{2}) \right] \) and \( M_2 = M_1 \), with probability at least \( 1 - 2\delta \), the following conditions hold.

1) \( \frac{D^1_T}{(1+\epsilon)M_1} \leq \frac{k^T}{n} \leq \frac{(1-\epsilon)k^T}{(1-\epsilon)M_1} \);
2) For every \( v \) such that \( I_v \geq k^T \), \( \frac{D^2(v)}{M_2} \geq (1-\epsilon)k^T \); and
3) For every \( v \) such that \( I_v \leq (1-2\epsilon)k^T \), \( \frac{D^2(v)}{M_2} \leq I_v + \epsilon k^T \).

Under these 3 conditions, obviously, if \( I_v \geq k^T \) then \( D^2_2(v) \geq \frac{(1-\epsilon)k^T}{M_2} \geq \frac{(1-\epsilon)k^T}{1+\epsilon} \). Thus, \( I_v \geq \frac{nD^2_2(v)}{M_2} \) and \( I_v \leq \frac{nD^2_2(v)}{M_2} \). Now we prove \( \frac{nD^2_2(v)}{M_2} \) is not much smaller than \( k^T \), which means when we use \( T \) as a filtering threshold, the influence spread of any false positive vertex is close to the true threshold \( t^T \). Let \( B = 4(\epsilon - 2) \ln \frac{n}{\delta} \), so \( D^1_T = \left[ Y_1(\epsilon, \frac{\delta}{2}) \right] \geq \frac{B(1+\epsilon)}{\epsilon^2} \).

When \( n \geq 1 \) and \( \delta \leq \frac{1}{4} \), \( B > 5 \). Thus,
\[
\frac{nT}{M_2} - \epsilon k^T = \frac{n(1-\epsilon)D^1_T}{(1+\epsilon)M_1} - \epsilon k^T \\
\geq \left[ \frac{D^1_T}{M_2} \right] (1-\epsilon)^2 - \epsilon k^T \\
\geq \frac{B(1+\epsilon)\epsilon}{1+\epsilon} (1-\epsilon)^2 - \epsilon k^T
\]

Since \( \frac{B(1+\epsilon)}{\epsilon^2} \geq 1 \), we have
\[
\frac{nT}{M_2} - \epsilon k^T \geq (1-\epsilon)^2 - \epsilon k^T \geq 1 - 2\epsilon - \epsilon k^T \geq 1 - 2\epsilon - \epsilon k^T
\]

Our algorithm returns the set of vertices \( S = \{ u | D_T(u) \geq T \} \). Summarize the above analysis, we have that with probability at least \( 1 - 2\delta \), (1) if \( I_u \geq k^T \), \( v \in S \), and (2) \( \min_{u \in S} I_u \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \).

Note that we prove \( \min_{u \in S} I_u \geq \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \) in Theorem 2 is just for showing there is a relative error bound. In practice, we use \( \left( 1 - \frac{\epsilon^2}{1+\epsilon} \right) - \epsilon k^T \) to calculate the upper bound of relative error, since it is tighter than \( 1 - \frac{\epsilon^2}{1+\epsilon} \).

**Remark.** One may ask why we do not apply Lemma 4 on \( R_1 \) and return all vertices \( u \) such that \( D_T(u) \geq T \). In such a case, we do not need \( R_2 \). However, we cannot do this because probabilistic support (implication) is not transitive [22]. Even when we know \( M_1 \geq \frac{n[Y_1(\epsilon, \frac{\delta}{2})]}{(1+\epsilon)^k} \) with high probability, we cannot establish the conditions (1) and (2) in Lemma 4 for \( R_2 \). Lemma 4 holds for \( \frac{n[Y_1(\epsilon, \frac{\delta}{2})]}{(1+\epsilon)^k} \) randomly generated RR sets without any prior knowledge, while we do have some prior knowledge about \( R_1 \) that \( D^1_T = \left[ Y_1(\epsilon, \frac{\delta}{2}) \right] \). To fix this issue, we generate \( R_2 \) that contains another \( M_2 = M_1 = |R_1| \) independent RR sets to find influential vertices. Then Lemma 4 holds for \( R_2 \).

To analyze the efficiency, we investigate the number of RR sets needed, since Section III-C already indicates that the maintenance cost for computing influence dynamically is proportional to the sample size. Corollary 3 also implies that, when \( \epsilon \leq \frac{1}{3} \), \( \delta \leq \frac{1}{4} \) and \( D^1_T = \left[ Y_1(\epsilon, \frac{\delta}{2}) \right] \), with probability at least \( 1 - \frac{(n+k+2)\delta}{2n} \), \( \frac{nD^2_2(v)}{(1+\epsilon)^k} \leq M_1 \). It is easy to verify that both sides of this inequality are \( \Theta(\frac{n\ln n}{\epsilon^2}) \). Thus, with high probability, \( M = |R| = M_1 + M_2 = \Theta(\frac{n\ln n}{\epsilon^2}) \). It is worth noting that, according to Dagum et al. [10], even when we know which vertex is the \( k \)-th most influential individual, to obtain an \( (\epsilon, \delta) \)-estimation of \( k^T \), at least \( \Theta(\frac{n\ln n}{\epsilon^2}) \) RR sets are needed. Considering normally \( k^T \ll n \), the minimum number of RR sets to achieve an \( (\epsilon, \delta) \)-estimation of \( k^T \) is \( \Theta(\frac{n\ln n}{\epsilon^2}) \), which is only smaller than our sample size \( \Theta(\frac{n\ln n}{\epsilon^2}) \) by at most a factor of \( \ln n \).
Comparison to [27] For the top-\(k\) tracking algorithm in [27], we set the absolute error \(\epsilon\) to the same value of the error in our method, which is less than \(4\epsilon P^k\) according to Theorem 2 (suppose magically we know the value of \(P^k\) beforehand). Then the number of RR sets is \(O(\frac{P^{k\min}k}{\epsilon^2})\), where \(P^k\) is the maximum individual influence. It is obvious that \(P^k \geq P^k\) and in real social networks, the gap between \(P^k\) and \(P^k\) can be large. Thus, our algorithm is normally more efficient than the top-\(k\) tracking algorithm in [27], when the errors controlled by the two methods are the same in absolute value.

C. Maintaining Ranks of Vertices and \(D^k_1\) Dynamically

By applying the Linked List structure [27] on \(R_2\), we dynamically maintain all vertices sorted by their degrees in \(R_2\). As demonstrated in Fig. 1, the vertices with the same degree in \(R_2\) are grouped together in a doubly linked list whose first node is a special node called a head node. All head nodes are sorted in a doubly linked list (the vertical one in Fig. 1). A nice property of the Linked List structure is that maintaining it does not increase the complexity of updating RR sets. For the interest of space, we skip the details of how to maintain the Linked List, which can be found in our previous work [27]. When a top-\(k\) individual query is issued, by the Linked List structure on \(R_2\), we only need \(O(k')\) time to return all vertices \(u\) such that \(D_2(u) \geq \frac{1-\epsilon}{1+\epsilon}D^k_1\), if there \(k'\) such vertices.

Besides dynamically maintaining ranks of vertices by their degrees in \(R_2\), we still need to maintain \(D^k_1\) efficiently and dynamically. This is because \(D^k_1\) is the signal used in deciding sample size. Moreover, we need the value of \(D^k_1\) in order to return all vertices \(u\) such that \(D_2(u) \geq \frac{1-\epsilon}{1+\epsilon}D^k_1\) when a top-\(k\) individual query is issued. Unfortunately, [27] did not discuss how to maintain \(D^k_1\).

To maintain the value of \(D^k_1\) against updates, we also apply the Linked List structure on \(R_1\). We record \(H_k\), the head node of the doubly linked list containing vertices whose degrees in \(R_1\) are all \(D^k_1\). We also need a bias \(b\) which indicates there are \(k-b\) vertices \(u\) such that \(D_1(u) > D^k_1\). Suppose due to an update, \(D_1(u)\) increases by 1. Let \(D_1(u)^{old}\) be the value before the update and \(D_1(u)^{new}\) the value after the update. Denote by \(H_k\) degree the degree of vertices that \(H_k\) is their head node and \(H_k\) number the number of such vertices. Let \(H_k\) up be the head node above it and \(H_k\) down the head node below it. To update \(H_k\) and \(b\), we have two cases depending on \(D_1(u)^{old}\).

Input: seed set size \(k\) and \(R\) (a collection of RR sets)
Output: \(S_k\)
1: \(S_k \leftarrow \emptyset\)
2: for \(i \leftarrow 1\) to \(k\) do
3: \(S_k \leftarrow \text{argmax}_{v \in V \setminus S_k} D(S_k \cup \{v\})\)
4: return \(S_k\)

Algorithm 3: Greedy Algorithm

1) If \(D_1(u)^{old} \neq H_k\) degree, we do not need to update \(H_k\) or \(b\) and
2) If \(D_1(u)^{old} = H_k\) degree, there are two subcases according to \(b\). If \(b = 1\) before the update, we set \(H_k\) as \(H_k\) up, and set \(b\) to the number of vertices in the linked list whose head is the updated \(H_k\). If \(b > 1\) before the update, we do not update \(H_k\) but we decrease \(b\) by 1.

Similarly, if \(D_1(u)\) of a vertex \(u\) decreases by 1, to update \(H_k\) and \(b\), we have three cases depending on \(D_1(u)^{old}\).

1) If \(D_1(u)^{old} < H_k\) degree or \(D_1(u)^{old} > H_k\) degree + 1, we do not need to update \(H_k\) and \(b\);
2) If \(D_1(u)^{old} = H_k\) degree + 1, we do not update \(H_k\) but we increase \(b\) by 1;
3) If \(D_1(u) = H_k\) degree, there are two subcases. If \(b = H_k\) number before the update, we set \(H_k\) as \(H_k\) down and set \(b = 1\). If \(b < H_k\) number before the update, we do not update \(H_k\) or \(b\).

Clearly, the updates on \(H_k\) and \(b\) only take \(O(1)\) time when \(D_1(u)\) changes by 1 for a vertex \(u\). After the maintenance, \(D_1 = H_k\) degree.

When a RR set is updated/inserted/deleted, \(D_1(u)\) changes at most by 1 for each \(u\). We need \(O(1)\) time to update its inverted index and only \(O(1)\) time to update the linked list data structure, \(H_k\) and \(b\). Thus, the linked list and our maintenance of \(D^k_1\) do not increase the complexity of updating RR sets. Also, \(D^k_1\) can be retrieved in \(O(1)\) time.

Moreover, if we set \(k = 1\), by employing the method described above, we can maintain \(D^1_1 = \max_{u \in V} D_1(u) = D^k_1\), which is used in deciding a proper sample size for maintaining RR sets against network updates for IM queries in Section V-A.

V. Efficient Influence Maximization Queries

In this section, we tackle a different version of the top-\(k\) influential vertex tracking task, that is, influence maximization (IM). Similar to Section IV, we first develop an efficient method to decide if we have a proper amount of RR sets. Then we propose efficient implementations of the greedy algorithm for IM queries on the maintained RR sets, which work well in practice, especially when the number of RR sets is small.

A. Algorithm and Sample Size

Like IM in static networks [2], [19], [23], [24], our goal is to achieve \(1 - \frac{1}{\epsilon} - \frac{1}{\epsilon}\) optimal IM queries on dynamic networks with high probability.

Unlike tracking top-\(k\) influential individuals, which is more in the flavor of a ranking problem, IM is a combinatorial optimization problem. Besides sampling a number of RR sets to approximate influence spreads of vertices or vertex sets, an IM query also needs to execute a greedy algorithm, such as...
Algorithm 3, on the sampled RR sets to find the best k-seed set. In the state-of-the-art static network IM algorithms [19], [24], the value of $D(S_k)$, where $S_k$ is returned by Algorithm 3, is used as a signal to decide if the current sample size of $\mathcal{R}$ is enough to achieve $1 - \frac{1}{2} - \varepsilon$ optimal IM queries with high probability. Unfortunately, for IM on dynamic networks, we cannot use $D(S_k)$ as the signal because running Algorithm 3 on $\mathcal{R}$ may take time linear to the size of $\mathcal{R}$ (the sum of numbers of vertices in all RR sets in $\mathcal{R}$), which is unaffordable for real-time updates.

Dinh et al. [11] proposed to use $D^* = \max_{u \in \mathcal{V}} D(u)$ as the signal to decide if we have enough RR sets for IM on static networks. Note that by using our Linked List structure, $D^*$ can be accessed in $O(1)$ time. The algorithm in [11] keeps sampling RR sets until $D^* = \left| \mathcal{Y}_1 \left( \frac{e}{\sqrt{1 - \frac{1}{\sqrt{e}} - \varepsilon}}, \delta \right) \right|$. The intuition of [11] and some other polling based algorithms [19], [24] is to make sure $\Pr\{D(S_k) \leq (1 + \varepsilon_1) I(S_k)\} \geq 1 - \delta_1$ and $\Pr\{\frac{M D(S_k)}{n} \geq (1 - \varepsilon_2) I(S_k)\} \geq 1 - \delta_2$, where $S_k$ is the seed set extracted by running the greedy algorithm on $\mathcal{R}$, and $S_1$ is the optimal $k$-seed set. In such a case, due to the submodularity of $D(\mathcal{S})$ with respect to $S$, we have $\Pr\{I(S_k) \geq \frac{1}{1 - (1 - \varepsilon_1) I(S_k)} \} \geq 1 - \delta_1$. Unfortunately the proof of $\Pr\{D(S_k) \leq (1 + \varepsilon_1) I(S_k)\} \geq 1 - \delta_1$ in [11] is incorrect because a union bound is missed.

We give a theoretically sound method to maintain RR sets, where the signal used for deciding sample size can be accessed in $O(1)$ time. First, for the maintained collection of RR sets $\mathcal{R}$, we also split it into two disjoint parts $\mathcal{R}_1$ and $\mathcal{R}_2$. Denote by $D_1(S)$ and $D_2(S)$, respectively, the degrees of $S$ in $\mathcal{R}_1$ and $\mathcal{R}_2$. Let $M_1 = |\mathcal{R}_1|$ and $M_2 = |\mathcal{R}_2|$. Let $N = \binom{n}{k_{max}}$, where $k_{max}$ is the maximum k of an IM query. Our algorithm works as follows,

1. Update the RR sets in $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ when the network structure changes using the method in [27].
2. Maintain the invariant that $D_1^*$, the greatest degree in $\mathcal{R}_1$, always equals $\left| \mathcal{Y}_1 \left( \frac{e}{\sqrt{1 - \frac{1}{\sqrt{e}} - \varepsilon}}, \delta \right) \right|$. Thus, with high probability, $D_1^*$ can be efficiently maintained against network updates and it can be accessed in $O(1)$ time when needed.

**Theorem 3.** Our algorithm returns a seed set $S_k$ for an IM query with $k \leq k_{max} \leq \frac{n}{2}$, such that $\Pr\{I(S_k) \geq \frac{1}{1 - \frac{1}{2} - \varepsilon} I(S_1^*)\} \geq 1 - \delta$, where $S_k^*$ is the optimal k-seed set.

**Proof:** Suppose $D_1(u) = D_1^*$, which means when we pick the vertex with maximum degree in $\mathcal{R}_1$, we get $u$. According to Lemma 1, by applying the union bound we have $\Pr\{I^* \geq \frac{1}{1 - (1 - \varepsilon_1) I(S_k^*)} \} \geq 1 - \frac{\delta}{2}$. Thus, we have lower bound of $I^* = \max_{u \in \mathcal{V}} I_u$ with high probability. Specifically, we have $\Pr\{I^* \geq \frac{n D_1(u)}{n D_1^*} \} \geq 1 - \frac{\delta}{2}$. Since $D_1^* = \left| \mathcal{Y}_1 \left( \frac{e}{\sqrt{1 - \frac{1}{\sqrt{e}} - \varepsilon}}, \delta \right) \right|$, with probability at least $1 - \frac{\delta}{2}$, $M_1 \geq \frac{4 \ln (e - 2) n \frac{\ln n}{\sqrt{\varepsilon}}}{\varepsilon^{\frac{1}{2}}}$. When $M_2 \geq \frac{4 \ln (e - 2) n \frac{\ln n}{\sqrt{\varepsilon}}}{\varepsilon^{\frac{1}{2}}}$, for any k-seed set $S$, applying Corollary 1 and utilizing the fact that $I(S_k^*) \geq I^*$ and $I(S_k^*) \geq I(S)$, we have

$$\Pr\{\forall S \subseteq V, |S| = k : \frac{D_2(S)}{M_2} - I(S) \leq \frac{\varepsilon I(S_k^*)}{n} \} \geq 1 - \frac{2 \delta}{3}$$

When $\frac{D_2(S)}{M_2} - I(S) \leq \frac{\varepsilon I(S_k^*)}{n}$ for all k-seed set $S$, we have

$$\frac{I(S_k)}{n} \geq \frac{D_2(S)}{M_2} - \varepsilon I(S_k^*) \geq \frac{(1 - \varepsilon)(1 - \varepsilon) I(S_k^*) - \varepsilon I(S_k^*)}{n} = \frac{1 - \varepsilon - (2 - 4 \varepsilon) I(S_k^*)}{n}.
$$

Therefore, applying the union bound, we have $\Pr\{I(S_k) \geq \frac{1}{1 - \frac{1}{2} - \varepsilon} I(S_k^*)\} \geq 1 - \delta$.

**Remark of mistakes in [11]** The major error made by [11] is that the union bound is missed when $\Pr\{\frac{M D(S_k)}{n} \leq (1 + \varepsilon_1) I(S_k)\}$. In [11], when the sampling phase ends, it is guaranteed that $D(S_k) \geq \left| \mathcal{Y}_1 \left( \frac{e}{\sqrt{1 - \frac{1}{\sqrt{e}} - \varepsilon}}, \delta \right) \right|$, where $\varepsilon_1 = \frac{\sqrt{1 - (1 - \varepsilon) I(S_1^*)}}{1 - \varepsilon_1}$. Dinh et al. [11] made a claim that $D(S_k) \geq \left| \mathcal{Y}_1 \left( \frac{e}{\sqrt{1 - \frac{1}{\sqrt{e}} - \varepsilon}}, \delta \right) \right|$ implies $\Pr\{\frac{M D(S_k)}{n} \leq (1 + \varepsilon_1) I(S_k)\} \geq 1 - \frac{\delta}{2}$. However, we cannot directly apply Theorem 1 on $S_k$ to get this conclusion, because $S_k$ is deliberately picked by the greedy algorithm, where multiple candidate sets are involved. To better understand this issue, let us recall our proof of Lemma 2 and Lemma 3. In our proof, no matter the sampled RR sets are, we always look at every vertex one by one and then apply the union bound. But $S_k$ is deliberately picked by the greedy algorithm on $\mathcal{R}$, it is easy to find that $S_k$ depends on the sampled RR sets. This introduces extra uncertainty. Also, this issue is similar to the overfitting issue in machine learning [18]. We relate the sampled RR sets $\mathcal{R}$ to the training data, $S$ to a classifier and $\frac{M D(S_k)}{n}$ to $S$’s accuracy on $\mathcal{R}$. Then the parameter of a classifier $S$ is the vertices in $S$. The parameter learning process (the greedy algorithm) returns only one $S$ but it involves multiple other candidate sets. If the number of candidate sets (corresponds to the size of parameter space in machine learning) is huge, picking the parameters (a seed set $S_k$) that has a very high accuracy on the training data may lead to overfitting.

Based on Theorem 3, to achieve $(1 - 1/e - \varepsilon)$ optimal IM queries with probability at least $1 - \frac{\delta}{2}$ (similar to the quality guarantees in the state-of-the-art IM algorithms [19], [24] on static networks), under any seed set size $k \leq k_{max}$, we maintain the invariants $D_1^* = \left| \mathcal{Y}_1 \left( \frac{e}{\sqrt{1 - \frac{1}{\sqrt{e}} - \varepsilon}}, \delta \right) \right|$, and $M_2 = \frac{M_1 \ln \frac{\ln n}{\varepsilon}}{\ln \frac{\ln n}{\varepsilon}}$. Similar to tracking influential individuals, with probability at least $1 - \frac{\delta}{2}$, we have $M_1 = O\left( \frac{n \ln n}{\varepsilon^2} \right)$ and $M_2 = O\left( \frac{n \ln n}{\varepsilon^2} \right)$. Thus, with high probability, in total $M = |\mathcal{R}| = M_1 + M_2 = O\left( \frac{n \ln n}{\varepsilon^2} \right)$. Since the average number of edges and vertices traversed when generating an RR set is no more than $\frac{n}{M} = O\left( \frac{n \ln n}{\varepsilon^2} \right)$, we have the cost $C(\mathcal{R}) = O\left( \frac{k_{max} n \ln n}{\varepsilon^2} \right)$, where $C(\mathcal{R})$ is the total number of edges and vertices traversed when generating $\mathcal{R}$, which is in the same order of the correct bound of $C(\mathcal{R})$ in the method in [20] and also the bound in [24].

In practice, $O\left( \frac{k_{max} n \ln n}{\varepsilon^2} \right)$ is too large because it is linear to $k_{max}$ and $k_{max}$ may easily be in the order of hundreds. Thus, in our experiments, in stead of strictly implementing...
our algorithm, we sample fewer RR sets by ignoring the factor $k_{\text{max}}$. This is similar to [20] where $C(R)$ is set to $O \left( \frac{(m+n) \ln n}{\varepsilon^2} \right)$ by ignoring $k_{\text{max}}$ in the correct bound $O \left( \frac{k_{\text{max}}(m+n) \ln n}{\varepsilon^2} \right)$. Specifically, our practical solution is to just maintain the invariant $D^* = \frac{1}{|Y_1(\frac{1}{2-\sqrt{2}})}$ (we do not split $R$ into $R_1$ and $R_2$ in the practical solution). It is not difficult to derive that in this way the sample size $M$ is roughly $\frac{1}{\varepsilon^2}$ times the theoretically sound sample size $O \left( \frac{k_{\text{max}} \ln n}{\varepsilon^2} \right)$. We demonstrate in experiments that our practical solution normally leads to much fewer RR sets than maintaining $C(R) = 32(m+n) \ln n$ in [20], and the quality of seed set mined is not compromised.

### B. Speeding Up the Greedy Algorithm for IM Queries

In addition to fast maintenance of the RR sets, we also need to run the greedy algorithm to find seed vertices. Efficient implementation of the greedy algorithm becomes critical. In the following discussion, we assume that the greedy algorithm is conducted on a collection of RR sets $R$. Note that if one strictly implements our method as described in Section V-A, which has theoretical guarantees, then $R$ should be $R_2$.

Our Linked List data structure in Fig. 1 can be used to implement the greedy algorithm (Algorithm 3). Algorithm 4 shows the implementation. In line 1, we take $O(n)$ time to copy the Linked List $D$ to $D'$. This step is essential because $D$ cannot be modified during an IM query. If we modify $D$ when executing an IM query, when the query ends $D(u)$ may not equal the number of RR sets containing $u$ such that $D$ cannot be used in the greedy algorithm (Algorithm 3) for a new IM query. It is easy to see that the cost of the rest part (after line 1) of Algorithm 4 is $O(\sum_{i=1}^{M} |R_i|)$, where $|R_i|$ is the number of vertices in the $i$-th RR set in $R$. Thus, the computational cost of Algorithm 4 is $O(n + \sum_{i=1}^{M} |R_i|)$.

Ohsaka et al. [20] reported that employing the Lazy Evaluation techniques [16] can achieve better performance in practice. However, the method in [20] still needs to take $O(n)$ time to copy $D$ to $D'$. In large networks, if the maintained RR sets are not many, it is possible that $\sum_{i=1}^{M} |R_i|$ is even smaller than $n$, the number of vertices. In such a case, copying the whole $D$ may be a waste of time.

Intuitively, in the greedy algorithm, when $k$ is small, most vertices are useless because their degrees in $R$ are too small and probably they are never be picked as a seed. Exploiting this intuition, we design more efficient algorithms by only copying part of $D$ to $D'$. Specifically, we use a threshold $T_D$. When iterating $D$ from the top head node, $D(v)$ is copied to $D'$ only if $D(v) > T_D$. We show that this filtering strategy can achieve good performance with provable guarantees.

Suppose Algorithm 4 returns a seed set $S_k$, and $S'_k$ is the seed set returned by the greedy algorithm where only vertices $v$ such that $D(v) > T_D$ are copied to $D'$. Let $U = \{ v \mid D(v) > T_D \}$ and $L = \{ v \mid D(v) \leq T_D \}$. Suppose $S'_k = \{ u_1, \ldots, u_k \}$, where $u_j$ is the $j$-th seed added to $S_k$. Define $M(u_j; S_k) = D(\{u_1, \ldots, u_{j-1}\} \cup \{u_j\}) - D(\{u_1, \ldots, u_{j-1}\})$ the marginal gain of $u_j$. Suppose $u_q$ is the first seed added to $S_k$ such that $M(u_q; S_k) \leq T_D$. If such $q$ does not exist, we set $q = k + 1$.

**Theorem 4.** $D(S'_k) \geq D(S_k) - (k - q + 1)T_D$. $\Box$

**Proof:** According to the submodularity of $D(S)$ with respect to $S$, it is easy to find that $M(u_j; S_k) \leq D(u_j) - D(U)$ for every $i$. For $i \leq q - 1$, $D(u_i) \geq M(u_i) > T_D$. Thus, when running the greedy algorithm by copying the whole $D$, in the first $q - 1$ iteration of choosing seeds, $u_i$ is always the vertex with the maximum marginal gain in the $i$-th iteration, no matter only vertices in $U$ or all vertices in $V$ are considered. So we have that the first $q - 1$ seeds of $S_k$ and $S'_k$ are the same. Thus, $S_k = \{ u_1, u_2, \ldots, u_{q-1}, v_q, \ldots, v_k \}$, where $v_i$ could be different from $u_i$.

We prove that $M(v_j; S_k) \leq T_D$ by contradiction. If $M(v_j; S_k) > T_D$, then $D(v_j) > D(U)$ and $v_j$ should be considered in building $S'_k$. But for the $q$-th seed $u_q$, $M(u_q; S'_k) \leq D(\{u_1, u_2, \ldots, u_{q-1}\} \cup \{v_q\}) - D(\{u_1, u_2, \ldots, u_{q-1}\})$, which contradicts the fact that $u_q$ has the largest marginal gain in the $q$-th iteration of building $S'_k$.

Since $M(v_q; S_k) \leq T_D$, we have $D(S_k) = D(\{u_1, \ldots, u_{q-1}\}) + \sum_{j=1}^{q-1} M(v_j; S_k)$. According to the submodularity of $D(S)$ and the greedy algorithm, it is easy to verify that $M(v_j; S_k) \geq M(v_j; S_k)$ for $j \geq q$. Thus, we have $D(S_k) \leq D(\{u_1, \ldots, u_{q-1}\} \cup \{k - q + 1\} \cup \{v_q\}) \leq D(\{u_1, \ldots, u_{q-1}\}) + (k - q + 1)T_D$. Because $\{u_1, \ldots, u_{q-1}\} \subseteq S'_k$, we have $D(S'_k) \geq D(\{u_1, \ldots, u_{q-1}\}) \geq D(S_k) - (k - q + 1)T_D$. $\Box$

Apparently, when $q = k + 1$, we have $D(S'_k) = D(S_k)$. Also, by setting $T_D = \min \{ \frac{D'}{k^2}, D' - 1 \}$, we have $D(S'_k) \geq \frac{1}{2} D(S_k)$.

**Corollary 4.** If $D' \geq 2$ and $T_D = \min \{ \frac{D'}{k^2}, D' - 1 \}$, then $D(S'_k) \geq \frac{1}{2} D(S_k)$. $\Box$

**Proof:** First, $T_D = \min \{ \frac{D'}{k^2}, D' - 1 \}$ ensures that $u$ is always considered in building $S'_k$ if $D(u) = D'$. Also, $T_D = \min \{ \frac{D'}{k^2}, D' - 1 \} < D'$. So we have $q \geq 2$, where $t$ is the first time that a seed added to $S'_k$ has a marginal gain no greater than $T_D$. Thus, if $k = 1$, $D(S'_k) = D(S_k)$. When $k = 2$, $D(S'_k) \geq D' \geq 2D' \geq 2D'$. When $k \geq 3$, $T_D = \frac{D' \cdot (k - 1)}{k^2} \leq \frac{D'}{k^2}$ because $D' \geq 2$. Therefore, $\frac{D(S'_k)}{D(S_k)} \geq D(S'_k) \geq \frac{D(S'_k) - (k - q + 1)T_D}{D(S_k)} \geq \frac{D'}{k^2} \geq \frac{D}{k^2} = 1/2$. $\Box$

Based on our analysis, we propose Algorithm 5, the New Greedy Algorithm, which also exploits the lazy evaluation method for maximizing monotone and submodular functions. The iterated returns $T_D$ by Algorithm 5 can tell us if the seed set found has the same quality as the seed set returned by Algorithm 4. In our implementation, we set $T_D = \min \{ \frac{D'}{k^2}, D' - 1 \}$ and run
the New Greedy algorithm. If the returned $q$ is not $k + 1$, we reset $T_D$ to $-1$ and re-run the New Greedy algorithm. Setting $T_D = -1$ is equivalent to the query algorithm in [20]. By doing so we can guarantee that the returned seed set $S_k$ has $D(S_k)$ at least $(1 - 1/e)D(\hat{S}_k)$, where $D(\hat{S}_k) = \max_{|S| = k} D(S)$. In our experiments we find that $T_D = \frac{1}{e^3}$ is an effective and efficient threshold, because we never need to reset $T_D$ to re-run the New Greedy algorithm.

VI. EXPERIMENTS

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

A. Experimental Settings

We used five real network data sets that are publicly available online (http://snap.stanford.edu, http://www.cs.ubc.ca/~wlu/ and https://an.kais.ac.kr/traces/WWW2010.html). Table II shows the basic statistics of the five data sets.

To simulate dynamic networks, for each data set, we randomly partitioned all edges exclusively into 3 groups: $E_1$ (85% of the edges), $E_2$ (5% of the edges) and $E_3$ (10% 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 1. For each edge $(u,v) \in E_3$, we generated a weight increase update $\langle u,v,+,1,- \rangle$ (timestamps ignored at this time). For each edge $(u,v) \in E_2$, we generated one weight decrease update $\langle u,v, -, \Delta, - \rangle$ and one weight increase update $\langle u,v,+,\Delta, + \rangle$, where $\Delta$ 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 the update stream, and thus ran the experiments 10 times. Note that for the 10 instances, although the base networks and the 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, that is, the whole graph. We set $w_{uv} = \frac{1}{\text{in-degree}(v)}$, where 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 $\langle u,v,+,1,- \rangle$ (again, timestamps ignored at this time). For each edge $(u,v) \in E_2$, we generated one weight decrease update $\langle u,v, -, \Delta, - \rangle$ and one weight increase update $\langle u,v,+,\Delta, + \rangle$, where $\Delta$ 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 top-$k$ influential individuals, that is, the parameters in Theorem 2, we set $k = 50$, $\delta = 0.001$ and $\varepsilon = 0.1$, which means that the relative error rate is roughly bounded by $36.5\%$ according to the proof of Theorem 2. For influence maximization (IM), as illustrated at the end of Section V-A, we implement a practical solution that maintains $D^* = \left\lfloor \frac{1}{\varepsilon} \right\rfloor \left\{ Y_1 \left( \frac{\varepsilon}{1 - \varepsilon}; \frac{2}{\varepsilon^2} \right) \right\}$. We set $\varepsilon = 0.0025$ such that $1 - \frac{1}{\varepsilon} - (2 - \frac{1}{\varepsilon})\varepsilon = 0.5$. Remember that $1 - \frac{1}{\varepsilon} - (2 - \frac{1}{\varepsilon})\varepsilon$ is the approximation ratio of the theoretical sound algorithm in Section V-A which roughly maintains $k_{max}$ times the number of RR sets as that of the practical solution.

To mimic the real application environment, we inserted an IM query in the update stream every $\tau$ edge weight updates. The reason we did not insert top-$k$ individual queries is because outputting top individual vertices is very efficient (thanks to our Linked List data structure that maintains the ranking of vertices). To better simulate the queries of users, we inserted $\tau$ random IM queries for each $\tau$ edge weight updates. We report the ratio of $C(R)$
of this baseline to the $C(R)$ in our algorithm. Note that this ratio is approximately the ratio of the number of RR sets of the baseline to the number of RR sets maintained by our algorithm. We also compare the query algorithm of [20] (Lazy Evaluation) to our New Greedy algorithm.

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

B. Tracking Influential Individuals

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 [3], [8], [13], [14], [23], [24] always uses 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 vertices really matters here. Even 20,000 times MC simulations may not be able to distinguish vertices with close influence spread. As a result, the ranking of vertices may differ much from the true 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 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 vertices 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 (e.g., 10 days on the final snapshots of Flixster).

Tables III reports on the wiki-Vote and Flixster data sets the recall of the sets of influential vertices returned by our algorithms and the maximum error rates of the false positive vertices in absolute influence value. The results are obtained by taking the average of the results on the 10 runs on 10 instances. Our methods achieved 100% recall every time as guaranteed theoretically. Moreover, the true error rates were substantially smaller than the maximum error rate provided by our theoretical analysis.

For the other 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 vertices at the final snapshots of the 10 instances are similar, at least our algorithms are stable, that is, insensitive to the order of updates. The similarity between two sets of influential vertices is measure by the Jaccard similarity.

Fig. 2 shows the results where II, . . . , I10 represent the results of the first, . . . , tenth instances, respectively. ST denotes the result obtained by computing the influential vertices directly from the final snapshot using our sampling methods without any updates. Fig. 2 shows that the outcomes from different instances are very similar, and they are similar to the outcome from ST, too. The minimum similarity is over 90%.

2) Scalability & Comparison with [27]: We also tested the scalability of our algorithm and the top-k influential vertex tracking algorithm in [27]. Fig. 3 shows the average running time with respect to the number of updates processed, where “RE IC” (RE is short for relative error) is our algorithm under the IC model, while “AE IC” (AE is short for absolute error) stands for the algorithm in [27] under the IC model. The average is taken on the running times of the 10 instances. The point at #Updates=0 of each curve in Fig. 3 represents the time spent by sampling enough RR sets on the base network. In all cases, our algorithm handles the whole update stream in substantially shorter time than the baseline in [27]. Our algorithm under the LT model scales up roughly linearly. Under the IC model the running time sometimes increases more than linear. This is probably 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 this value is roughly 0.9 but becomes 1 finally. So some updates of the IC model may lead to big change of the maximum influence or the average influence, and consequently the running time increases more than linearly.

We also demonstrate the limitations of controlling $\max_{u \in S}D_k - D_u$ as an absolute error, where $S$ is the set of vertices mined. Fig. 4 shows how the value $D_k$ varies over time, and so does the theoretically maximum error $\max_{u \in S}D_k - D_u$ over time. The value of $\frac{D_k}{n}$ is estimated by $\frac{\sum D_k}{n}$. Note that for the algorithm in [27], we set $\varepsilon_1$ to the same for both the IC and the LT models, where the parameter $\varepsilon_1$ is exactly the maximum absolute error $\max_{u \in S}D_k - D_u$ we want to control. The maximum error of our algorithm (RE Error) is either only a little bigger or smaller than the maximum error of the baseline (AE Error). Moreover, we find that the value of $\frac{D_k}{n}$ varies over time, especially under the IC model. In Fig. 4 (c), (d) and (e), sometimes the error of [27] (AE Error) is even greater than $\frac{D_k}{n}$, which makes the result meaningless because [27] will return all vertices as influential vertices. This demonstrates the limitation of [27], which controls an absolute error.

<table>
<thead>
<tr>
<th>TABLE III. RECALL AND MAXIMUM ERROR RATE. THEORETICAL VALUES HOLD WITH HIGH PROBABILITY.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>wiki-Vote</td>
</tr>
<tr>
<td>Flixster</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV. RUNNING TIME (s) ON STATIC NETWORKS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
</tr>
<tr>
<td>wiki-Vote</td>
</tr>
<tr>
<td>Flixster</td>
</tr>
<tr>
<td>soc-Pokec</td>
</tr>
<tr>
<td>flicker-growth</td>
</tr>
<tr>
<td>Twitter</td>
</tr>
</tbody>
</table>
We also report the running time of ST, which is directly applying our sampling methods without any updates on the final snapshot of each dataset to extract influential individuals. Table IV compares the running time of ST and the total time (denoted by "Total") that our algorithm samples RR sets on the base network and deals with the whole update stream. In Table IV, the running time of Total is at most 40 times longer than that of ST. Thus, if we re-sample RR sets from scratch every time when the network updates, we probably can only deal with tens of updates within the same time as Total spends on all updates. However, the number of total updates is huge, tens of thousands or even hundreds of millions. This indicates that the non-incremental algorithm (re-sampling RR sets from scratch when the network updates) is not competitive at all.

C. Influence Maximization

1) Effectiveness of Sample Size: We demonstrate that our practical solution that maintains \( D^* = \frac{1}{\epsilon} \log \left( \frac{1 - \epsilon}{1 - \epsilon \log n} \right) \) is effective by testing results of IM queries on the final snapshot of each data set. The algorithm that collects a large number of RR sets until \( C(\mathcal{R}) = 32(m + n) \log n \) in [20] and DSSA [19], the state-of-the-art IM algorithm on static networks, are compared. The parameters of DSSA are set such that DSSA is 0.5-optimal with probability \( 1 - \frac{1}{2^k} \). The three algorithms in comparison decide the sample size in different ways and generate different number of RR sets. Comparing effectiveness of the three algorithms is actually comparing effectiveness of the decisions of sample size in the three algorithms. Thus, we use \( "D^* = \frac{1}{2} \log \left( \frac{1 - \epsilon}{1 - \epsilon \log n} \right)" \) and \( "C(\mathcal{R}) = 32(m + n) \log n" \) to denote our practical solution and the method in [20], respectively. We did not run the method \( "C(\mathcal{R}) = 32(m + n) \log n" \) on the twitter data set because it is too costly.

To extract the seed set \( S_k \) of the final snapshot, we set \( T_D = \min \{ \frac{1}{\epsilon^2}, D^* - 1 \} \) and ran the New Greedy algorithm (Algorithm 5) on the RR sets generated by each algorithm. It turned out that by setting \( T_D = \min \{ \frac{1}{\epsilon^2}, D^* - 1 \} \) the New Greedy Algorithm always returned \( S_k \) such that \( D(S_k) \geq (1 - \frac{\epsilon}{2}) \max S \in g D(S) \). Algorithm 4 and Lazy Evaluation (the query algorithm in [20]) were also tested as the IM query algorithm, but the results were all very similar. Thus, we only report the results by using New Greedy as the IM query algorithm.

To measure the effectiveness of a seed set \( S_k \), we generated another collection of RR sets \( \mathcal{R}' \) such that the cost \( C(\mathcal{R}') \) is \( 32(m + n) \log n \) (except on Twitter dataset C(\mathcal{R}') is set to \( (m + n) \log n \) because \( 32(m + n) \log n \) is too costly). The influence of \( S_k \) is estimated using \( \mathcal{R}' \). By Theorem 1, we checked \( D'(S_k) \), the degree of \( S_k \) in \( \mathcal{R}' \), and found that \( \mathcal{R}' \) estimated \( I(S_k) \) accurately. We calculate that, with high probability, the relative error rate is at most 2%. Fig. 5 shows the results where all values are averages taken on the results from 10 instances. All algorithms have pretty close performance except that DSSA is slightly worse than the others. This is because DSSA is dedicated for a specific seed set size \( k \) and it always generates a smaller number of RR sets than other methods. This demonstrates that our practical solution that maintains \( D^* = \frac{1}{2} \log \left( \frac{1 - \epsilon}{1 - \epsilon \log n} \right) \) is effective in practice, although it does not have theoretical guarantees.

2) Scalability: Fig. 6 reports the running time of maintaining RR sets for influence maximization queries over update streams. Similar to tracking influential individuals, our algorithm for maintaining RR sets scales roughly linearly under the LT model, while under the IC model in some cases it does not. Still the probable reason is our experiment setting. The maximum and the average influence do not vary much against updates under the LT model, while some updates under the IC model lead to big changes of the maximum influence or the average influence. For the largest dataset Twitter, our algorithms processed 0.3 billion updates in less than 4 hours.

To compare with the method in [20], we report the ratio of the number of RR sets needed by the method in [20] to the number of RR sets maintained by our algorithm. This ratio reflects the improvement in efficiency of our algorithm over [20] because the cost of maintaining RR sets against an update and the cost of an influence maximization query are both roughly linear to the number of RR sets. The results are shown in Fig. 7. Our algorithm consistently maintains fewer RR sets than [20] as the ratio is consistently greater than 1. On soc-Pokec and the largest data set Twitter, our improvement is more than an order of magnitude. The results on the two small data sets are similar and are omitted limited by space. One may find that for the LT model, the ratio in Fig. 7 increases over the update stream in general, while for the IC model, in two data sets the ratio keeps decreasing. This is due to the experimental setup of updates under the LT and IC model. The effects of update streams on influence spreads of vertices are different under the LT and IC models.

We also report the efficiency of IM query algorithms (implementations of the greedy algorithm) in Fig. 8. New Greedy (Algorithm 5), LinkedList Greedy (Algorithm 4), Lazy Evaluation (the IM query algorithm in [20]) and DSSA [19] are compared. New Greedy, LinkedList Greedy and Lazy Evaluation were all ran directly on the maintained (by our practical solution that keeps \( D^* = \frac{1}{2} \log \left( \frac{1 - \epsilon}{1 - \epsilon \log n} \right) \)) RR sets of the final snapshot, while DSSA first sampled a number of RR sets based on the final snapshot, and then extracted a seed set by running the greedy algorithm on the sampled RR sets. Limited by space, we omit the results on the two small data sets, which are similar to Fig. 8. The results show that our New Greedy algorithm is always the fastest query algorithm, and the larger a network, the bigger the improvement of the New Greedy algorithm over the baselines. For the largest data
set Twitter, our New Greedy algorithm returns a seed set with good quality within 300ms, and it is an order of magnitude faster than the Lazy Evaluation algorithm in [20], and two to three orders of magnitude faster than the DSSA algorithm [19] running on the static network.

VII. CONCLUSION

In this paper, we tackled two versions of tracking top- \( k \) influential vertices in dynamic networks. We adopted two simple signals to decide a proper number of RR sets for the two tasks, and showed that with high probability our sample size ensures that the result has good quality guarantees. We reported a series of experiments on five real networks and demonstrated the effectiveness and efficiency of our algorithms.

Parallelizing our methods in large distributed systems is an interesting future direction. Since our solutions are based on independent sampling, they have a great potential to be parallelized for further accelerations. Also, for influence maximization task, devising methods that decide the sample size efficiently and only keep a small number of RR sets to handle big seed set size \( k \) with quality guarantees still remains an open problem.

REFERENCES

Fig. 6. Scalability (Influence Maximization).

Fig. 7. Comparison to $32(m+n)\log n$.

Fig. 8. Influence Maximization Query Time.