Tracking Influencers in Decaying Social Activity Streams With Theoretical Guarantees

Junzhou Zhao, Pinghui Wang, *Senior Member, IEEE*, Wei Zhang, Zhaosong Zhang, Maoli Liu, Jing Tao, and John C. S. Lui, *Fellow, IEEE*

***Abstract*— Influence maximization (IM) is the fundamental problem in many real world applications such as viral marketing, political campaign, and network monitoring. Although extensively studied, most studies on IM assume that social influence is static and they cannot handle the *dynamic influence challenge* in reality, i.e., a user's influence is varying over time. To address this challenge, we formulate a novel influencer tracking problem over a social activity stream. In order to keep the solutions up-to-date and forget outdated data in the stream smoothly, we propose a *probabilistic-decaying social activity stream (PDSAS)* model that enforces each social activity in the stream participating in the analysis with a probability decaying over time. Built on the PDSAS model, we propose a family of streaming optimization algorithms to solve the influencer tracking problem. SIEVEPAIT can identify influencers from a special kind of probabilistic addition-only social activity streams with high efficiency, and guarantees an $(1/2-\epsilon)$ approximation ratio. BASICIT leverages SIEVEPAIT as a building block to identify influencers from general PDSASs, and also guarantees an $(1/2-\epsilon)$ approximation ratio. HISTIT improves the efficiency of BASICIT, and still guarantees an $(1/4-\epsilon)$ approximation ratio. Experiments on real data show that our methods can find high quality solutions with much less computational costs than baselines.**

***Index Terms*— Influence maximization, streaming optimization, online social networks.**

I. INTRODUCTION

IN FLUENCER marketing has become an essential strategy for companies to leverage the social influence of influencers on online social networks (OSNs) to promote product sales. For example, the Amazon Influencer Program [1] allows the influencers from Twitter, Facebook, YouTube, and Instagram to showcase the products and recommend to their followers. It is estimated that the influencer marketing industry is worth up to $15 billion by 2022 [2]. Besides its usage in promoting sales, influencer marketing is also used by authorities to fight against vaccine lies during the COVID-19 pandemic [3] and to spread rumors during the Russia-Ukraine war [4]. The key tech behind influencer marketing is *influence maximization (IM)*, which aims to select k seed users, i.e., influencers, such that they can jointly trigger the maximum influence spread on a network. Since the seminal works of Domingos et al. [5] and Kempe et al. [6], IM has been extensively studied for decades.

IM relies on estimating the influence spread of a set of seed users on a network based on users' pairwise influence probabilities, which, however, are usually unknown, and need to be inferred from *social activities* observed on OSNs [7], [8], [9]. For example, in Twitter, if a user u tweets a tweet containing a hashtag, and later, his follower v also tweets a tweet containing the same hashtag (or v simply retweets u's tweet), then it implies that user u influenced user v. If such activities are frequently observed, then u reflects strong influence on v. After pairwise influence probabilities of users in an OSN are obtained, there have been extensive studies on efficiently solving the IM problem [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]. Most of these studies assume that the pairwise influence probabilities as well as the social network under study are static, in other words, social influence is static. However, in practice, social influence could be dynamic driven by the evolving social activities in an OSN. For example, a Twitter user v frequently retweeted another user u's tweets in the past few weeks, but stopped retweeting recently because u posted offensive content and v unfollowed u, therefore u cannot influence v anymore. Indeed, Myers and Leskovec [20] reported that the Twitter network is highly dynamic with about 9% of all connections changing in every month. It is therefore unrealistic to assume that social influence is static, otherwise the influencers identified today may just quickly become outdated tomorrow. This raises the *influencer tracking* problem that we want to address: *in an OSN with evolving social activities, how to efficiently identify the influencers at any query time, as illustrated in Fig. 1?*

A straightforward way to handle the dynamic social influence challenge is that we always re-estimate the pairwise influence probabilities at every time we need to query influencers in the OSN. Obviously, this approach will incur too much computational overhead which may be unaffordable if we need to frequently perform the influencer query. In the literature, there have been some efforts trying to efficiently address the dynamic social influence challenge such as the heuristic approaches [21], [22], constructing updatable sketches [23], [24], and the interchange greedy method [25]. However, these methods either do not have theoretical guarantees on the solution quality such as the heuristic approaches, or they cannot handle the highly dynamic and fast evolving social

Manuscript received 4 May 2022; revised 22 February 2023 and 13 July 2023; accepted 17 September 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Chen. Date of publication 16 October 2023; date of current version 18 April 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62272372, Grant 61902305, and Grant U22B2019. *(Corresponding author: Pinghui Wang.)*

Junzhou Zhao, Pinghui Wang, Wei Zhang, Zhaosong Zhang, and Jing Tao are with the MOE KLINNS Laboratory, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: phwang@mail.xjtu.edu.cn).

Maoli Liu and John C. S. Lui are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR.

Digital Object Identifier 10.1109/TNET.2023.3323028

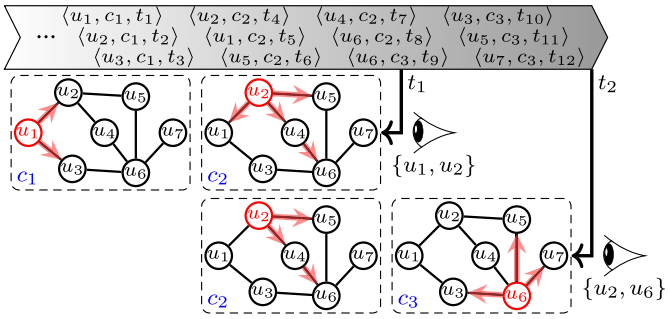


Fig. 1. Influencer tracking over decaying social activity streams.

activity data, e.g., the interchange greedy method actually degrades to the re-computation approach when influencers change significantly over time. In addition, the computational complexities of these methods are still too high to handle large-scale OSNs with millions of users that may generate a large amount of social activities in every second [26].

In this work, we propose a streaming optimization approach to efficiently address the dynamic social influence challenge in IM. Our approach allows to track influencers from the evolving social activity data in near real-time with guaranteed solution quality. As social activities are actually the direct evidences of users' influence in an OSN, it is better to directly take social activities as inputs in the influencer tracking algorithm. In practice, social activities are often generated in a streaming fashion, forming a high-speed and endless *social activity stream*, as illustrated in Fig. 1. It is thus desired to find streaming algorithms that only access each social activity in the stream just once. Moreover, in order to keep the identified influencers *fresh*, we also want the streaming algorithm to be able to gradually forget the outdated social activities in the stream. One approach is to leverage the popular sliding-window stream model [27] where only the most recent social activities participate in the analysis and all past social activities outside of the window are completely discarded. However, the sliding-window stream model has the weakness that it abruptly forgets all past social activities outside of the window, and some of which may be still important. Hence, it is not a smooth manner to forget outdated data in the stream. As a result, the sliding-window stream model may break *continuity* over time, and result in undesirable solutions. Let us consider a motivating example below.

Suppose we want to identify influencers on Twitter based on users' retweeting activities. Alice has been an influencer for years, but recently she is sick and does not post new tweets, and hence the number of retweets from her followers drops significantly. If we apply the sliding-window stream model, then only a few social activities related to Alice are observed and Alice will be mistaken for non-influential, even though she has been influential for years and her absence is just temporal.

The above example motivates us to find better data stream models that are able to forget old data in the stream in a smoother manner than the sliding-window stream model. To this end, we propose the *Probabilistic-Decaying Social Activity Stream* (PDSAS) model. The PDSAS model has a feature that social activities from past to present all have a chance to participate in the analysis but recent social activities have a larger chance than past social activities. As time advances, each social activity participates in the analysis with a decaying probability. Hence, the PDSAS model can overcome the weakness of the sliding-window stream model, and preserve solution freshness and continuity simultaneously.

Built on the PDSAS model, we design a family of streaming optimization algorithms, namely SIEVEPAIT, BASICIT, HISTIT, and HISTIT-RED, that can identify near-optimal influencers from the social activity stream in near real-time. First, SIEVEPAIT is designed to identify influencers from a special kind of *probabilistic addition-only social activity streams*, in which each social activity participates in the analysis with a constant probability. Then, BASICIT leverages SIEVEPAIT as a basic building block to identify influencers over general probabilistic-decaying social activity streams. Finally, HISTIT is designed to significantly improve the efficiency of BASICIT, and HISTIT-RED is a practical implementation of HISTIT. Importantly, we theoretically show that our approach can find near-optimal solutions with both time and space efficiency.

In summary, our contributions are as follows:

- We formulate a novel problem of influencer tracking over probabilistic-decaying social activity streams, in which the old data in the stream is forgotten in a smoother manner than the sliding-window stream model. This formulation ensures that the solutions of the problem are both fresh and continuous as time advances (see Section II).
- We propose a family of streaming optimization algorithms, namely SIEVEPAIT, BASICIT, HISTIT, and HISTIT-RED. Our algorithms guarantee a constant $(1/2-\epsilon)$ approximation ratio (for BASICIT), and an $(1/4-\epsilon)$ approximation ratio (for HISTIT and HISTIT-RED) for faster speed (see Sections V and VI).
- We conduct extensive experiments on various real-world data streams to validate the effectiveness of our algorithms. The results demonstrate that our approach outputs high quality solutions with much less computational costs than baselines (see Section VII).

II. PROBLEM FORMULATION

We consider a social network $G=(V,E)$ where V is a set of users (or nodes) and $E \subseteq V \times V$ is a set of user relations (or edges). Edges in the network may be undirected like the friend relations in Facebook, or directed like the following relations in Twitter. In the latter case, let edge (u,v) denote that user v follows user u, indicating the potential that u can influence v. The frequently used symbols are listed in Table I.

A. Some Definitions

Users in a social network could generate *social activities* such as retweeting tweets on Twitter, commenting or replying to posts on Reddit, and purchasing products on Amazon. Formally, let $a \triangleq \langle u, c, t\rangle$ denote a social activity representing that user u performed a specific activity c (e.g., purchasing an

TABLE I
FREQUENTLY USED SYMBOLS

Symbol	Description
$G = (V, E)$	a graph with node set V and edge set E
$a = \langle u, c, t\rangle$	a social activity, i.e., user u did activity c at time t
$\sigma(u; A) \subseteq V$	user u's influence spread given social activities A
$f(\cdot)$	the utility function
$p_t(a) \in [0, 1]$	participation probability of social activity a at time t
$h(\cdot)$	the decay function
$\mathcal{S}_t$	a stream of social activities by time t
$S, S_t, S_t^* \subseteq V$	a set of users (at time t)
$F_t(\cdot)$	surrogate objective function
$l_a^{(i)}$	the i-th lifetime sample of social activity a
$I(a), I_l(a)$	I-set of social activity a (at time $t_a + l$)
$\delta(u\|S), \Delta(u\|S)$	utility gain of adding node u to set S
$\mathcal{A}, \mathcal{A}_l$	a SIEVEPAIT instance, or the utility of its output

iPhone SE) at time t. Social activities from an OSN are direct evidences reflecting one user's influence on another [8], [9], [28]. For example, consider two social activities $a_1 = \langle u, c, t\rangle$ and $a_2 = \langle v, c, t + \delta\rangle$ performed by two users u and v at time t and $t + \delta$, respectively. If $(u, v) \in E$ and δ is no larger than a threshold, then it is likely that user v performed the same activity c due to the influence of user u. Hence, if a user's social activities trigger many follow-up social activities in the network, then the user is considered to be influential. Formally, we define the pairwise user influence as follows.

Definition 1 (Pairwise Influence): Given a set of social activities A generated by users in network G, we say that user u influenced user v, denoted by $u \overset{A}{\rightsquigarrow} v$, if v is reachable from u in G and there exist social activities $\langle u, c, t\rangle$ and $\langle v, c, t'\rangle \in A$ such that $t' - t \leq \delta$ for some threshold δ.

It is worthy noting that, in the above definition, we do not require $\langle u, c, t\rangle \in A$ but assume $\langle u, c, t\rangle$ is always observed when considering the influence of u. This is because our goal is to evaluate user u's influence based on the observed social activities performed by the other users, and in many cases, the original social activity that triggers the observed social activity is already known. For example, in Twitter and Weibo, the original tweet retweeted by the observed tweets is usually attached as a reference.

We further define a user's *influence spread* as follows.

Definition 2 (Influence Spread): Given a set of social activities A generated by users in network G, a user u's influence spread, denoted by $\sigma(u; A)$, is a set of users who were influenced by u with respect to A, i.e., $\sigma(u; A) \triangleq \{v \mid u \overset{A}{\rightsquigarrow} v\}$.

In other words, the influence spread of a user is simply the set of users directly or indirectly influenced by the user based on the observed social activities. We will also assume that $u \in \sigma(u; A)$. For example, in Fig. 2, for social activities A in network G, we first construct the *diffusion graphs* of two activities c_1 and c_2, respectively. Then, the influence spread of each user can be derived straightforwardly.

We further define the *influence spread of a user set* as the union of each individual user's influence spread, i.e., $\sigma(S; A) \triangleq \bigcup_{u \in S} \sigma(u; A)$. To evaluate the *influence of a user set*, we use a function $f\colon 2^V \mapsto \mathbb{R}_{\geq 0}$ to map the influence spread to a non-negative real number, e.g., the simplest one could be $f(\sigma(S; A)) \triangleq |\sigma(S; A)|$. Here, we only require that f is a *monotone submodular function*, which is commonly used to capture the diminishing returns property [29]. Specifically,

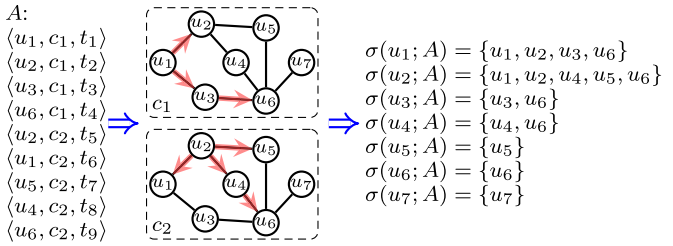


Fig. 2. Influence spread of each user based on observed social activities A. Note that because $t_9 - t_7 > \delta$, hence u_5 did not influence u_6.

we say that f is *monotone* if for all subsets $X \subseteq Y \subseteq V$, it holds that $f(X) \leq f(Y)$, and f is *submodular* if for all $X \subseteq Y \subseteq V$ and $v \in V \backslash Y$, it holds that $f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y)$, i.e., the additional benefit of an element v is no larger when added to set Y than added to set X (please refer to [30] for a nice survey of submodular function and its applications).

B. Probabilistic-Decaying Social Activity Streams

An OSN could continuously generate a stream of social activities and form an endless *social activity stream*. If we treat each social activity in the stream equally, then many outdated social activities will be used as evidences to infer each user's influence spread, and this may result in that many identified influencers were influential before but not now, i.e., the solution is not fresh. An alternative approach is that we only consider the most recent social activities within a time window, aka the sliding-window stream model [27]. However, as we discussed previously, the sliding-window stream model abruptly discards all past data outside of the window which is not a smooth manner to forget outdated data in the stream, and may result in inappropriate results.

To address these limitations, we propose a *probabilistic-decaying social activity stream* (PDSAS) model to represent the social activity stream. The PDSAS model has a feature that each social activity in the stream has a chance to participate in the analysis but recent social activities have a larger chance than historical social activities (see Fig. 1). In this manner, solution freshness can be guaranteed, and meanwhile, outdated social activities are forgotten gradually in a smooth manner.

Formally, let $\mathcal{S}_t \triangleq \{a \mid t_a \leq t\}$ denote the set of all occurred social activities by time t, where t_a is the timestamp of social activity a. At time t, for each $a \in \mathcal{S}_t$, we let a participate in the analysis with probability $p_t(a) \triangleq h(t - t_a)$, and $h\colon \mathbb{Z} \mapsto [0, 1]$ is a non-increasing decay function that assigns a social activity with age x a participation probability $h(x)$. By choosing proper decay functions, the PDSAS model can capture many special settings. For example, if we choose $h(x) = 1, \forall x$, then the stream will never decay; if we choose $h(x) = 1$ for $x \leq W$ (where W denotes the window size) and $h(x) = 0$ otherwise, then the PDSAS model becomes the sliding-window stream model. In practice, we prefer to choose h to be strictly decreasing so that each social activity in $\mathcal{S}_t$ participates in the analysis with a decreasing probability as it ages and hence gradually fades, e.g., $h(x) \propto \exp(-\lambda x)$. In short, the PDSAS model is a general stream model that allows to forget outdated data in a smooth manner.

Remark: The decay function could also be designed to be element-specific, i.e., the decay function for social activity a is $h_a(x)$. Or, the decay function could be a function of an social action's category or other attributes. This allows the PDSAS model to decay inhomogeneously.

C. The Influencer Tracking Problem

Based on the PDSAS model, we now formulate a novel influencer tracking problem.

Problem 1 (Influencer Tracking): Given a network G and a social activity stream $\mathcal{S}_t$ with decay function h, at any query time t, we want to find a subset $S_t^* \subseteq V$ containing at most k users and maximizing the expected influence on G, i.e.,

$$S_t^* = \underset{S\subseteq V \wedge |S|\leq k}{\arg\max}\ \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]. \tag{1}$$

Here, $\sigma(S;\mathcal{S}_t,h)$ denotes the influence spread of set S with respect to the PDSAS model, and the expectation is taken upon the randomness that each social activity participates in the analysis with a probability.

It is noteworthy to mention that, the decay function and utility function in together govern the selection of users in the network. Solutions that are fresh and jointly have large influence are likely to be good solutions, and this feature is indeed desired in many real-world applications.

Remark: For ease of explaining our idea, in what follows, we will mainly focus on the exponential decay function $h(x) = p_0 e^{-\lambda x}$ where $0 \leq p_0 \leq 1$ and $\lambda \geq 0$. Note that our framework can also be generalized to other general decay functions.

III. Constructing A Surrogate Objective

Given a user set $S \subseteq V$, we find that it is prohibitive to compute the exact value of $\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$ because we have to consider the participation possibility of each social activity in the stream, and this will result in an exponential number of combinations, i.e., $O(2^{|\mathcal{S}_t|})$. To address this challenge, we propose a Monte-Carlo method to fast estimate $\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$ and reduce the complexity to $O(n)$ using n samples.

Inspired by the idea of *possible-world* method [31], we define a *realization* of a probabilistic decaying social activity stream as a sequence of social activities that actually participate in the analysis in one experiment. If we use $\mathcal{S}$ to denote a realization of $\mathcal{S}_t$, then $\mathcal{S} \subseteq \mathcal{S}_t$. Let $\mathbb{S}_t \triangleq \{\mathcal{S} \mid \mathcal{S} \subseteq \mathcal{S}_t\}$ denote the set of all realizations, i.e., all possible worlds. Because each social activity independently participates in the analysis, then each realization is observed with probability

$$P(\mathcal{S} \in \mathbb{S}_t) = \prod_{a\in\mathcal{S}_t} p_t(a)^{\mathbf{1}(a\in\mathcal{S})}\big(1-p_t(a)\big)^{\mathbf{1}(a\notin\mathcal{S})} \tag{2}$$

where $\mathbf{1}(\cdot)$ denotes the indicator function with $\mathbf{1}(\text{true}) = 1$ and $\mathbf{1}(\text{false}) = 0$.

For example, in Fig. 3, we show some realizations of the original stream, and each social activity $a \in \mathcal{S}_t$ is independently included in a realization with probability $p_t(a)$. Because recent social activities have a higher probability to participate in the analysis than historical social activities, hence recent social activities are more likely to be included in a realization.

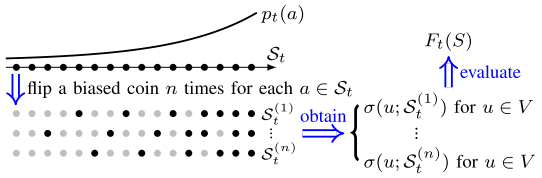


Fig. 3. Stream realizations and evaluating $F_t(S)$ from samples.

We will also refer to a realization as a *sample* of the stream. The Monte-Carlo method [32] states that a collection of samples independently drown from an identical distribution can provide an unbiased estimate of the expectation. Specifically, we have the following result.

Lemma 1: Let $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$ be a collection of n samples independently drown from distribution $P(\mathcal{S})$ in Eq. 2. Then,

$$F_t(S) \triangleq \frac{1}{n}\sum_{i=1}^{n} f(\sigma(S;\mathcal{S}_t^{(i)})) \xrightarrow{a.s.} \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))] \tag{3}$$

as $n \to \infty$, where $\xrightarrow{a.s.}$ denotes the almost-sure convergence.

Given a sample $\mathcal{S}_t^{(i)}$, which is actually a collection of social activities, we can calculate the influence spread $\sigma(s;\mathcal{S}_t^{(i)})$ for each user $s \in S$ (recall the example in Fig. 2). Therefore, we can easily obtain $\sigma(S;\mathcal{S}_t^{(i)})$ and hence $F_t(S)$. Intuitively, $\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$ is the average influence of set S on $|\mathbb{S}_t|$ stream realizations. $F_t(S)$ is the average influence of set S on n i.i.d. stream samples. By the Law of Large Numbers, $F_t(S)$ converges to $\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$ almost surely. Therefore, by choosing a sufficiently large sample size n, $F_t(S)$ will be a good approximation of $\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$. The Monte-Carlo method can reduce the computational complexity to just n evaluations of f, or we say n *oracle calls* and one oracle call refers to one evaluation of f. The sample size n can be determined by applying the Hoeffding's inequality.

Lemma 2: Given a set S and n i.i.d. samples $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$, if $n \geq \frac{\ln(2/\delta)}{2\epsilon^2\rho^2}$ where $\epsilon,\delta \in (0,1)$ and $\rho \triangleq \frac{\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]}{f(\sigma(S;\mathcal{S}_t))}$, then

$$\big|F_t(S) - \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]\big| < \epsilon\mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$$

holds with probability at least $1-\delta$.

In the following discussion, we will assume that sample size n is sufficiently large so that $F_t(S) \approx \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]$. Therefore, our goal becomes to find a set $S \subseteq V$ at each query time t to maximize the surrogate objective $F_t(S)$. Importantly, we find that $F_t(S)$ has the following property.

Lemma 3: F_t is a monotone submodular function.

Therefore, the influencer tracking problem boils down to two sub-problems: (i) how to fast generate stream samples $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$, and (ii) how to optimize the surrogate objective $F_t(S)$ in a streaming fashion. In the following discussions, we consider efficiently addressing these two sub-problems.

IV. Stream Sampling Methods

In this section, we design sampling methods to generate samples $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$ from the original stream $\mathcal{S}_t$, given decay function $h(x) = p_0 e^{-\lambda x}$, thereby solving sub-problem (i).

A. Fast Generating Stream Samples

We can use the approach illustrated in Fig. 3 to obtain a sample $\mathcal{S}_t^{(i)}$ from $\mathcal{S}_t$ straightforwardly. That is, at each time t, we sequentially scan the social activities in the stream; for each social activity $a \in \mathcal{S}_t$, we include a in sample $\mathcal{S}_t^{(i)}$ with probability $p_t(a)$ and ignore a with probability $1 - p_t(a)$. To obtain n samples, we simply scan the original stream n times, or just flip a biased coin n times for each social activity. The main weakness of this approach is that, as time advances from t to $t+1$, as each $p_t(a)$ decreases to $p_{t+1}(a)$, we have to rescan the stream. Obviously, this approach is not a streaming method and is inefficient.

In fact, given decay function $h(x) = p_0 e^{-\lambda x}$, we can sample the stream incrementally and obtain $\mathcal{S}_t^{(i)}$ from $\mathcal{S}_{t-1}^{(i)}$ in a more efficient manner. Specifically, for each social activity $a \in \mathcal{S}_{t-1}^{(i)}$, we still keep a in $\mathcal{S}_t^{(i)}$ with probability $e^{-\lambda}$, and for each newly arrived a at time t, we add a in $\mathcal{S}_t^{(i)}$ with probability p_0. The following lemma states that the obtained sample $\mathcal{S}_t^{(i)}$ indeed follows the desired distribution $P(\mathcal{S})$.

Lemma 4: Let $\mathcal{S}_t^{(i)}$ be a stream sample via incremental sampling at time t, then $P(a \in \mathcal{S}_t^{(i)}) = p_t(a)$ for each $a \in \mathcal{S}_t$.

Observe that, in incremental sampling, if a social activity is "alive" at time t, then it remains alive at time $t+1$ with probability $e^{-\lambda}$. Once the social activity is removed from the sample, it no longer participates in the analysis. Thus each social activity can be thought to have a *lifespan* which is the number of time steps the social activity remains alive and is included in the sample. As time advances, a social activity's lifespan decreases, and when the lifespan becomes zero, the social activity is discarded from the sample. We find that the initial lifespan of a social activity a, denoted by l_a, actually follows the following distribution

$$P(l_a = l) = \begin{cases} 1 - p_0, & l = 0, \\ p_0 e^{-\lambda(l-1)}(1 - e^{-\lambda}), & l = 1, 2, \ldots \end{cases} \quad (4)$$

When $p_0 = 1$, the distribution is actually the *geometric distribution* with parameter $e^{-\lambda}$.

Based on the above observation, we can obtain samples $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$ as follows. For each newly arrived social activity a, we sample n initial lifespans for a by Eq. 4, denoted by $l_a^{(i)}$, $i = 1, \ldots, n$, and $a \in \mathcal{S}_t^{(i)}$ if and only if $l_a^{(i)} > t - t_a$, i.e., the remaining lifespan at time t is positive.

Lifespan sampling is a more convenient way to sample $\mathcal{S}_t$ than previous methods, and it is suitable in the streaming setting. In addition, lifespan sampling inspires us to introduce the I-set representation of a social activity, which turns out to be very useful in later discussions.

B. Representing Social Activities by I-Sets

Motivated by lifespan sampling, we introduce *I-set*, a notation to conveniently represent each social activity in the PDSAS model.

Definition 3 (I-set): Given samples $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$, the I-set of a social activity a at time $t = t_a + l$ (i.e., l time steps after its arrival) is

$$I_l(a) \triangleq \{i \mid a \in \mathcal{S}_t^{(i)}\} = \{i \mid l_a^{(i)} > l\}. \quad (5)$$

i	$l_a^{(i)}$
1	2
2	7
3	5

l	$I_l(a)$	l	$I_l(a)$
0	$\{1,2,3\}$	4	$\{2,3\}$
1	$\{1,2,3\}$	5	$\{2\}$
2	$\{2,3\}$	6	$\{2\}$
3	$\{2,3\}$	7	$\emptyset$

Fig. 4. Generating I-sets from lifespan samples. Assume we have sampled $n = 3$ lifespans, i.e., 2, 7, and 5, for a social activity a (Left). Then by Eq. 5, we can generate the I-sets for social activity a (Right).

In other words, the I-set of a social activity is the set of indexes of stream samples that still include this social activity. Note that at time $t = t_a + l$, a social activity a is still included in sample $\mathcal{S}_t^{(i)}$ if $l_a^{(i)} > l$. As time advances, i.e., l increases, fewer samples still include a, hence the size of set $I_l(a)$ will decrease, or we say that the I-set shrinks over time. In fact, I-set is an equivalent way to characterize the decaying nature of a social activity in the PDSAS model. Specifically, as time advances, the probability that each social activity participates in the analysis decreases; equivalently, its I-set shrinks over time. If a social activity no longer participates in the analysis, then equivalently, its I-set becomes empty. Therefore, I-set is another way to represent the decaying social activity in the PDSAS model.

Using lifespan sampling, we can easily construct the I-sets of a social activity, i.e., we first sample n lifespan samples by Eq. 4, and then obtain I-sets by Eq. 5. An example is illustrated in Fig. 4.

Remark: As a special case, consider $p_t(a) = p_a \in [0, 1]$, i.e., each social activity participates in the analysis with a constant probability. Then each social activity's I-set will also be a constant and not shrink. We thus simplify the notation to $I(a)$. $I(a)$ can be directly obtained by flipping a biased coin (with probability p_a of heads) n times, and $I(a) \triangleq \{i\colon$ the i-th outcome is a head$\}$.

V. A Strawman Algorithm

Based on the proposed stream sampling methods and the I-set notation, we next present a basic strawman algorithm, called BasicIT, to address the influencer tracking problem. We show that our original problem can be reduced to a special influencer tracking problem where social activities do not decay. Using the algorithm developed for this special problem as a building block, we design BasicIT to solve the original problem.

A. Probabilistic Addition-Only Influencer Tracking

In the simplest case, the decay function is a constant, i.e., $h(x) = p_0$. So $p_t(a) = h(t - t_a) = p_0$ is time invariant. In this case, each social activity participates in the analysis with a constant probability, i.e., no decay. We refer to this special influencer tracking problem as the *Probabilistic Addition-only Influencer Tracking* (PAIT) problem. As its name suggests, in PAIT, social activities are only probabilistically added in and no social activity is removed from *a pool of active social activities* that are used to evaluate each user's influence spread on G. This unique feature will help us to find efficient streaming algorithms to solve the PAIT problem.

For each social activity $a \in \mathcal{S}_t$, because $p_t(a) = p_0$ is a constant, then a's I-set is also a constant — a special case that we have discussed in Section IV-B. Let $I(a)$ denote this

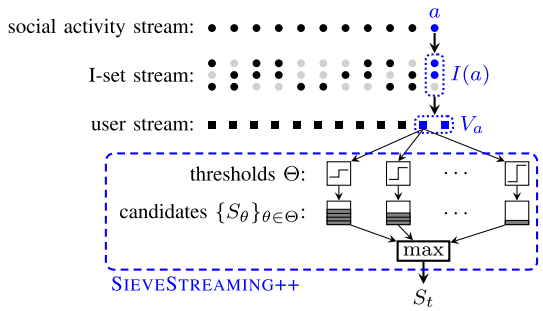


Fig. 5. Probabilistic addition-only influencer tracking. Each social activity is associated with a constant I-set. Due to the addition of social activity a in stream samples, the influence spreads of users in $V_a \subseteq V$ change.

constant I-set, which can be easily obtained by flipping a biased coin n times. Then, the original social activity stream can be represented as an I-set stream, as illustrated in Fig. 5.

By the definition of I-set, a social activity a is included in $\mathcal{S}_t^{(i)}$ for all $i \in I(a)$. In the Monte-Carlo framework, each user $u \in V$ is associated with n influence spreads evaluated on n samples respectively, i.e., $\sigma(u; \mathcal{S}_t^{(i)})$ for $i = 1, \ldots, n$. We say that a user's influence spread changes if at least one of these n influence spreads change. Let $V_a \subseteq V$ denote the set of users whose influence spreads change due to the addition of social activity a. In PAIT, because social activities are only added into samples $\{\mathcal{S}_t^{(i)}\}_{i=1}^n$, each user's influence spread will only increase as time advances. Therefore, when a arrives, we get affected users V_a and obtain a user stream, as shown in Fig. 5.

Hence, the PAIT problem becomes finding k users from the user stream to maximize the submodular objective F_t, which is actually a *streaming submodular optimization* (SSO) problem. Fortunately, SSO is well studied for several stream models [33], [34], [35], [36], [37]. In our case, the user stream can be viewed as an insertion-only stream because users are only added in and no user is removed from the stream. For the insertion-only SSO problem, the state-of-the-art streaming algorithms are SIEVESTREAMING [33] and its improvement SIEVESTREAMING++ [37]. Below, we briefly introduce SIEVESTREAMING and then show how to leverage SIEVESTREAMING++ to solve the PAIT problem.

The insertion-only SSO problem aims to select k elements from an insertion-only stream to maximize a monotone submodular function such that each element in the stream can only be accessed once. The high-level idea of SIEVESTREAMING is as follows. For each element in the stream, we calculate its *gain of utility* when adding it to the current solution. If the gain is no less than an *optimal threshold*, the element is added in the solution; otherwise ignored. This process continues until the budget is exhausted. In practice, the optimal threshold is unknown in advance. SIEVESTREAMING cleverly uses a set of thresholds and maintains a candidate solution for each threshold. Finally, the best candidate is returned as the solution which is guaranteed to be $(1/2-\epsilon)$-approximate. By dynamically updating the thresholds, SIEVESTREAMING++ further reduces the space complexity to $O(k/\epsilon)$.

To leverage SIEVESTREAMING++ to solve the PAIT problem, for each user u in the user stream, we need to calculate the gain of utility with respect to a user set S, denoted by $\Delta(u|S)$. It follows that

$$\begin{aligned}\Delta(u|S) &\triangleq F_t(S \cup \{u\}) - F_t(S) \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(f(\sigma(S \cup \{u\}; \mathcal{S}_t^{(i)})) - f(\sigma(S; \mathcal{S}_t^{(i)}))\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\delta^{(i)}(u|S)\end{aligned}$$

where $\delta^{(i)}(u|S)$ denotes the gain of utility evaluated on the i-th sample. We can then leverage SIEVESTREAMING++ to solve the PAIT problem (see Fig. 5). We refer to this algorithm as SIEVEPAIT, and its pseudo-code is given in Algorithm 1.

Algorithm 1 SIEVEPAIT

Input: social activity stream, budget k, and $\epsilon \in (0, 1)$
Output: influencers S_t at each query time t

$\Delta \leftarrow 0, \theta_{\min} \leftarrow \emptyset$, and $\text{LB} \leftarrow 0$;
foreach *social activity a arrived at time t* **do**
  $V_a \leftarrow$ users whose influence spreads change;
  // Lines 4-7 maintain a set of thresholds
  $\Delta \leftarrow \max\{\Delta, \max_{u \in V_a} F_t(\{u\})\}$;
  $\theta_{\min} \leftarrow \max\{\text{LB}, \Delta\}/2k$;
  Discard all sets S_θ with $\theta < \theta_{\min}$;
  $\Theta \leftarrow \{(1+\epsilon)^i \mid (1+\epsilon)^i \in [\theta_{\min}/(1+\epsilon), \Delta]\}$;
  // Lines 8-12 filter users by thresholds
  foreach $u \in V_a$ **do**
    foreach *threshold* $\theta \in \Theta$ **do**
      if $|S_\theta| < k$ *and* $\Delta(u|S_\theta) \geq \theta$ **then**
        $S_\theta \leftarrow S_\theta \cup \{u\}$;
        $\text{LB} \leftarrow \max\{\text{LB}, F_t(S_\theta)\}$;
  // Return S_t when a query is performed
  $S_t \leftarrow \arg\max F_t(S_\theta)$;

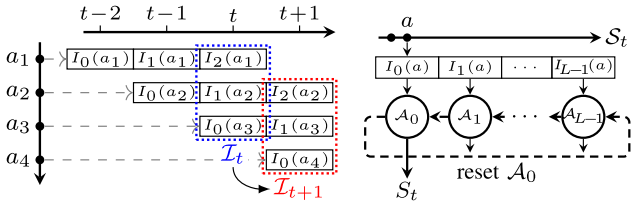
However, we notice that PAIT has two major differences with the insertion-only SSO. In the insertion-only SSO problem, each element appears only once in the stream, and the objective function is time invariant. While in our case, a user may appear multiple times in the user stream, and objective F_t is time-varying. Therefore, we still need to show that SIEVEPAIT in Algorithm 1 guarantees a constant approximation ratio. Leveraging the observation that each user's influence spread does not decrease in PAIT, we find that SIEVEPAIT indeed guarantees a constant approximation ratio.

Theorem 1: SIEVEPAIT guarantees an $(1/2-\epsilon)$ approximation ratio.

Let γ denote the average size of set V_a for all a. Then SIEVEPAIT has the following complexity.

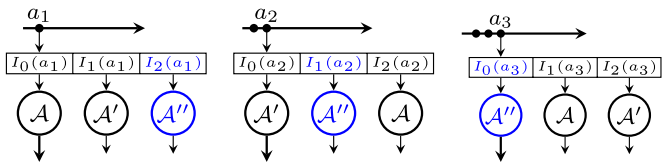
Theorem 2: SIEVEPAIT requires $O(\gamma n \epsilon^{-1} \log k)$ oracle calls to process each arrival social activity, and $O(k\epsilon^{-1})$ space to store the candidate solutions.

Remark: In the what follows, we treat SIEVEPAIT as a blackbox, whose input is a stream of constant I-sets $\{I(a) \mid a \in \mathcal{S}_t\}$, and output is an $(1/2-\epsilon)$-approximate solution of the PAIT problem. We use $\mathcal{A}$ to denote an instance of SIEVEPAIT, and we use SIEVEPAIT as a building block to design a strawman algorithm to address our original influencer tracking problem.



(a) Evolving I-sets ($L = 3$) (b) BASICIT algorithm

Fig. 6. Illustration of evolving I-sets and the BASICIT algorithm.



(a) Processing a_1 (b) Processing a_2 (c) Processing a_3

Fig. 7. Applying BASICIT to the stream in Fig. 6(a). Note that after processing a_3, $\mathcal{A}''$ has processed I-sets $I_2(a_1)$, $I_1(a_2)$, and $I_0(a_3)$, i.e., $\mathcal{I}_t$.

B. The BASICIT *Algorithm*

We now consider the influencer tracking problem under the general PDSAS model. In this case, each social activity participates in the analysis with a probability decreasing over time, and its I-set will shrink from $I_0(a)$ at time t_a, to $I_1(a)$ at time $t_a + 1$, and finally to an empty set, indicating that the social activity no longer participates in the analysis, as illustrated in Fig. 4. To simplify our discussion and motivate a basic streaming algorithm, we assume that each social activity can remain alive for at most L time steps, i.e., the lifespan is upper bounded by L. Therefore, each social activity will have at most L non-empty I-sets, and $I_l(a) = \emptyset$ for $l \geq L$. This could simplify our algorithm design. Note that we will remove this assumption later.

First, let us understand the evolving of I-sets in a stream using an example in Fig. 6(a). In Fig. 6(a), when social activity a_1 arrives at time $t-2$, its I-set is $I_0(a_1)$. At the next time step, a_1's I-set shrinks to $I_1(a_1)$, and meanwhile, a_2 arrives with I-set $I_0(a_2)$. At time t, a_3 arrives with I-set $I_0(a_3)$, and the I-sets of a_1 and a_2 shrink to $I_2(a_1)$ and $I_1(a_2)$, respectively. This process then continuous, and when an I-set becomes empty, the corresponding social activity no longer exists.

Let $\mathcal{I}_t \triangleq \{I_l(a) \neq \emptyset \mid a \in \mathcal{S}_t \wedge t_a + l = t\}$ be a family of non-empty I-sets at time t, formed by currently alive social activities. For example, in Fig. 6(a), $\mathcal{I}_t = \{I_2(a_1), I_1(a_2), I_0(a_3)\}$. In fact, at time t, we can treat $\mathcal{I}_t$ as a family of constant I-sets of social activities with constant participation probabilities. For example, in Fig. 6(a), at time t, we can think that a_1 participates in the analysis with an instantaneous constant probability $p_t(a_1)$ and its I-set is $I_2(a_1)$; a_2 participates in the analysis with an instantaneous constant probability $p_t(a_2)$ and its I-set is $I_1(a_2)$; and so on. Thus a simple idea of solving the influencer tracking problem is to feed I-sets in $\mathcal{I}_t$ to a SIEVEPAIT instance. Then the instance's output will be an $(1/2-\epsilon)$-approximate solution at time t. The challenge is that I-sets in $\mathcal{I}_t$ are evolving over time, and we can access each social activity only once. We cannot afford to process $\mathcal{I}_t$ again at each time $\mathcal{I}_t$ changes. Therefore, we need to find a clever way to process $\mathcal{I}_t$ in a streaming fashion.

We propose a simple algorithm, called BASICIT, to address this challenge under the assumption that lifespans are upper bounded by L. First, recall that when a social activity arrives, we can sample n lifespans and easily obtain its L I-sets (see Section IV-B). Next, we can maintain L SIEVEPAIT instances, denoted by $\{\mathcal{A}_l\}_{l=0}^{L-1}$, and use a "processing-and-shifting" scheme to process these L I-sets in parallel. Specifically, as illustrated in Fig. 6(b), BASICIT works as follows.

- **Processing.** Feed $I_l(a)$ to $\mathcal{A}_l$ for $l = 0, \ldots, L-1$.
- **Shifting.** Reset $\mathcal{A}_0$ and circularly shift these L SIEVEPAIT instances one unit to the left. We also need to relabel each instance so that the first one always starts from subscript 0.

We repeatedly execute the above two steps whenever a social activity arrives. We find that $\mathcal{A}_0$ always processed all the I-sets in $\mathcal{I}_t$, thus $\mathcal{A}_0$'s output will be the solution at time t. The pseudo-code of BASICIT is given in Algorithm 2.

Algorithm 2 BASICIT

```
Initialize L SIEVEPAIT instances {A_l}_{l=0}^{L-1};
foreach social action a do
    Obtain I-sets I_l(a), l = 0, ..., L − 1;
    for l = 0, ..., L − 1 do Feed A_l with I_l(a);
    S_t ← output of A_0;
    for l = 1, ..., L − 1 do A_{l−1} ← A_l;
    Reset A_0 and A_{L−1} ← A_0;
```

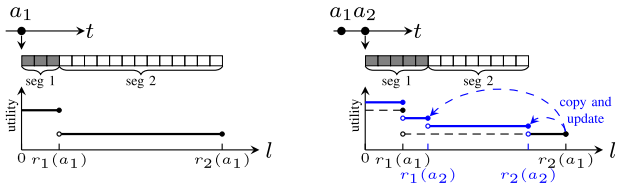
In Fig. 7, we use an example to explain the execution of BASICIT. Assume $L = 3$, and we maintain three SIEVEPAIT instances, namely $\mathcal{A}$, $\mathcal{A}'$, and $\mathcal{A}''$. When a_1 arrives, we obtain its I-sets and feed them to the three instances, respectively. Then $\mathcal{A}$ is reset, and they are all shifted to the left, and continuously process a_2. Note that the first instance always processed all the I-sets of current social activities in the stream. For example, after a_3 is processed, $\mathcal{A}''$ has processed I-sets $I_2(a_1)$, $I_1(a_2)$, and $I_0(a_3)$ which are exactly the I-sets in $\mathcal{I}_t$ (see Fig. 6(a)). Hence, $\mathcal{A}''$'s output is the correct solution.

It is straightforward to see that BASICIT has the same approximation ratio as SIEVEPAIT. While the update complexity and space complexity of BASICIT are both L times larger than SIEVEPAIT because L SIEVEPAIT instances are maintained.

Theorem 3: BASICIT guarantees an $(1/2-\epsilon)$ approximation ratio.

Theorem 4: BASICIT requires $O(L\gamma n\epsilon^{-1}\log k)$ oracle calls to process each social activity and needs $O(Lk\epsilon^{-1})$ space to maintain candidate sets.

Remark: BASICIT is only suitable for the case that lifespan upper bound exists and is small. This happens when the decay function has a fast decay rate. Otherwise L is large, and BASICIT needs to maintain a large number of SIEVEPAIT instances, and this will incur high CPU and RAM overloads. Therefore, BASICIT has a bottleneck for handling slowly decaying social activity streams. To address this bottleneck, we propose a fast method for influencer tracking.



(a) processing a_1 (b) processing a_2

Fig. 8. Example of using few SIEVEPAIT instances.

VI. A FAST ALGORITHM FOR INFLUENCER TRACKING

In this section, we propose HISTIT, to address the drawback of BASICIT. HISTIT also removes the assumption that lifespan is upper bounded. In addition, HISTIT is significantly faster than BASICIT with a little loss on solution quality but still ensures a constant approximation ratio.

The idea of HISTIT is to reduce the number of SIEVEPAIT instances in BASICIT. Intuitively, if just a few SIEVEPAIT instances are maintained, then the efficiency should be improved a lot. Specifically, we want to select a subset of SIEVEPAIT instances indexed by $\mathbf{l}_t \triangleq \{l_1^{(t)}, l_2^{(t)}, \ldots\}$, where $l_i^{(t)} \in \{0, 1, \ldots\}$ is an instance index, to approximate the rest at any time t. Roughly speaking, this can be thought of as using a histogram to approximate a curve, thus called HISTIT. In what follows, we elaborate two ways to reduce the number of SIEVEPAIT instances in BASICIT.

A. Grouping I-Sets Into Segments

We observe that, when generating the I-sets of a social activity using n lifespan samples, no matter how large each lifespan is, there are always at most n distinct non-empty I-sets. Consider the example in Fig. 4, we can see that $I_0(a) = I_1(a)$, $I_2(a) = I_3(a) = I_4(a)$, and $I_5(a) = I_6(a)$. That is, social activity a has only 3 distinct non-empty I-sets even though the lifespan could be as large as 7. Since consecutive I-sets are often the same, we group consecutively same I-sets into one segment. There will be at most n segments for each social activity, and each segment corresponds to a unique I-set. Denote the i-th segment of a by interval $[l_i(a), r_i(a)]$ for $1 \le i \le n$. For the example in Fig. 4, there are 3 segments, i.e., $[0, 1]$, $[2, 4]$, and $[5, 6]$, corresponding to 3 distinct I-sets, respectively.

The observation of I-set segmentation can be used to reduce the number of SIEVEPAIT instances in BASICIT. The idea is that if several SIEVEPAIT instances are fed with the same I-sets, then their outputs must be the same, and hence we only need to maintain just one of them. Specifically, suppose a is the first social activity in the stream arrived at time t and we have obtained its I-sets $\{I_l(a)\}_{l\ge 0}$. Because $I_l(a)$ will be fed to $\mathcal{A}_l$ and I-sets belong to the same segment are same to each other, hence $\mathcal{A}_l$ will have same output for l in the same segment. Therefore, we only need to maintain one SIEVEPAIT instance for each segment, say, those indexed by $\mathbf{l}_t = \{r_i(a)\}_{1\le i\le n}$. For example, in Fig. 8(a), because a_1 has two segments, we only need to maintain two instances, i.e., $\mathcal{A}_l$ for $l \in \{r_1(a_1), r_2(a_1)\}$.

When social activities keep arriving, we can update the current maintained SIEVEPAIT instances and create more instances based on current maintained instances. Specifically, consider a social activity a arrived at time t, and let $\mathbf{l}_{t-1}$ denote the set of indexes of maintained instances before processing a. For each segment $[l, r]$ of a, if $r \notin \mathbf{l}_{t-1}$, we can create $\mathcal{A}_r$ by copying the successor instance $\mathcal{A}_{l^*}$ where $l^* = \min\{l \mid l > r \wedge l \in \mathbf{l}_{t-1}\}$; if no such successor exists, we create a new instance as $\mathcal{A}_r$. We add r in $\mathbf{l}_{t-1}$ and obtain $\mathbf{l}_t$. Finally, for each $l \in \mathbf{l}_{t-1}$, we also need to feed $\mathcal{A}_l$ with the I-set of the segment that l belongs to.

For example, in Fig. 8(b), a_2 arrives at time t, and $\mathbf{l}_{t-1} = \{r_1(a_1), r_2(a_1)\}$. Because $r_1(a_2) \notin \mathbf{l}_{t-1}$ and $r_2(a_2) \notin \mathbf{l}_{t-1}$, we hence create two instances $\mathcal{A}_{r_1(a_2)}$ and $\mathcal{A}_{r_2(a_2)}$ by copying the same successor $\mathcal{A}_{r_2(a_1)}$. Then we feed segment one's I-set to $\mathcal{A}_{r_1(a_1)}$ and $\mathcal{A}_{r_1(a_2)}$, and feed segment two's I-set to $\mathcal{A}_{r_2(a_2)}$. We can verify that each SIEVEPAIT instance indeed correctly processed the I-sets.

Note that the first maintained instance still outputs the solution with the approximation ratio same to BASICIT. The pseudo-code of the algorithm using I-sets segmentation, i.e., HISTIT-SEG, is given in Algorithm 3.

Algorithm 3 HISTIT-SEG

```
1  l ← ∅;
2  foreach social action a do
3      Obtain segments [l_i(a), r_i(a)], 1 ≤ i ≤ n;
4      foreach segment [l, r] do Process(l, r);
5      S_t ← output of A_{l_1};
6      if l_1 = 0 then Kill A_{l_1}, and l ← l\{l_1};
7      for l ∈ l do A_{l-1} ← A_l, l ← l − 1;
8  Procedure Process(l, r)
9      if r ∉ l then
10         if r has no successor then create a new A_r;
11         else create A_r by copying its successor;
12         l ← l ∪ {r};
13     Feed the segment's I-set to A_i, i ∈ [l, r];
```

Remark: Leveraging I-set segments, we can reduce the number of SIEVEPAIT instances; however, as social activities keep arriving, we may need to keep creating SIEVEPAIT instances, resulting in a large number of instances to maintain. Hence, HISTIT-SEG may still suffer from high CPU and RAM overloads. To address this issue, we propose another way to further reduce the number of SIEVEPAIT instances.

B. Reducing Redundancy

Intuitively, if consecutive SIEVEPAIT instances have very close outputs, implying that they found similar solutions, then we think that they are redundant to each other, and maintaining only one of them is enough. Therefore, our second idea to reduce the number of running SIEVEPAIT instances in BASICIT is to remove those redundant ones. Our method is inspired by the idea of *smooth histogram* for analyzing sliding-window streams [38], [39]. To materialize this idea, we need to strictly define redundancy. To simplify our notations, we will use $\mathcal{A}_l$ to refer to the instance's output utility.

Definition 4: Given a small real number $\epsilon \in (0, 1)$, if $\mathcal{A}_j \ge (1-\epsilon)\mathcal{A}_i$ for $j > i$, then we say that instances with indexes between i and j are ϵ-redundant.

That is, since $\mathcal{A}_j$ and $\mathcal{A}_i$ are already close to each other, then instances between them are considered to be redundant.

Thus, we can regularly check the output utilities of SIEVEPAIT instances during the running of HISTIT-SEG (e.g., after processing each social activity), and kill those redundant ones. We will use HISTIT-RED to refer to the algorithm that further leverages the idea of reducing redundancy to improve efficiency.

However, there is one issue when practically implementing HISTIT-RED. That is, when creating an instance from its successor in Line 11 of HISTIT-SEG, if the successor of $\mathcal{A}_r$ were killed at some previous step (because $\mathcal{A}_r$ were redundant), then at current time, the maintained successor is actually not the true successor of $\mathcal{A}_r$. Creating $\mathcal{A}_r$ by copying the fake successor may cause issues. We postpone the solution to this issue, but simply assume that their is an "oracle" that can correctly create the SIEVEPAIT instance.

C. Theoretical Analysis of HISTIT-RED

The reducing redundancy operation in HISTIT-RED will slightly harm the solution quality, but we can theoretically show that the loss in solution quality can be bounded and HISTIT-RED still guarantees a constant approximation ratio.

To analyze the approximation ratio, we need to introduce some notations. Due to shifting and relabeling, $l \in \mathbf{l}_t$ and $l+1 \in \mathbf{l}_{t-1}$ actually index the same SIEVEPAIT instance, i.e., $\mathcal{A}_l$ at time t and $\mathcal{A}_{l+1}$ at time $t-1$ are the same SIEVEPAIT instance. In general, $l' \in \mathbf{l}_{t'}$ and $l \in \mathbf{l}_t$ index the same instance if $t' \leq t$ and $l' = l + t - t'$. If so, we say that $\mathcal{A}_{l'}$ at time t' is an ancestor of $\mathcal{A}_l$ at time t. To simplify the notations, we will always use $\mathcal{A}'$ to refer to an ancestor instance of $\mathcal{A}$. We find that HISTIT-RED maintains a histogram satisfying the following property.

Lemma 5: Let $\mathcal{A}_{l_i}$ and $\mathcal{A}_{l_{i+1}}$ be two consecutive SIEVEPAIT instances at time t. There are only two possible cases that could cause these two instances to be consecutive at time t.

Case 1 There is no social activity having a segment $[l, r]$ with r falling between the indexes of these two instances since these two instances were created.

Case 2 At some time $t' < t$, it holds that $\mathcal{A}'_{l_{i+1}} \geq (1-\epsilon)\mathcal{A}'_{l_i}$, and from time t' to time t, no instance is created between these two instances.

Lemma 5 actually states the two possible scenarios that two maintained instances are consecutive: either the two instances are consecutive since they were created (Case 1), or they become consecutive after redundant instances between them were removed (Case 2). Based on Lemma 5, we can obtain the approximation guarantee of HISTIT-RED.

Theorem 5: HISTIT-RED guarantees an $(1/4-\epsilon)$ approximation ratio.

The proof of Theorem 5 is rather complicated and the outline is as follows. If $l_1 = 0$, then it means that $\mathcal{A}_0$ exists. Because $\mathcal{A}_0$ processed all the I-sets in $\mathcal{I}_t$ as is in BASICIT, then the output is $(1/2-\epsilon)$-approximate. Otherwise, we consider l_1 and its most recently expired predecessor l_0. Since they were consecutive at the time l_0 did not expire, then by Lemma 5, there are two cases. If Case 1 holds, we can argue that $\mathcal{A}_{l_1}$ still completely processed the I-sets in $\mathcal{I}_t$, thereby returning a $(1/2-\epsilon)$-approximate solution. If Case 2 holds, then there exists some time $t' < t$ s.t. $A'_{l_1} \geq (1-\epsilon)A'_{l_0}$. This finally leads to an $(1/4-\epsilon)$-approximate solution. We provide the complete proof in the Appendix.

Finally, we find that the number of maintained SIEVEPAIT instance in HISTIT-RED is $O(\epsilon^{-1}\log k)$. We thus obtain the update time and space complexity of HISTIT-RED.

Theorem 6: HISTIT-RED requires $O(\gamma n \epsilon^{-2}\log^2 k)$ oracle calls to process each arrival social activity and needs $O(k\epsilon^{-2}\log k)$ space to maintain candidate solutions.

D. A Practical Implementation of HISTIT-RED

HISTIT-RED relies on an oracle that can correctly create $\mathcal{A}_r$, i.e., Line 11 in Algorithm 3. A naive way to implement this oracle is that: when an instance needs to be killed, we swap it out of the RAM, and store the unprocessed I-sets corresponding to it; when $\mathcal{A}_r$ needs to be restored, we swap it in the RAM and feed it with the unprocessed I-sets. Obviously, it is inefficient. Instead, we propose a faster and practical implementation of HISTIT-RED.

The idea is that, we do not wish to accurately restore $\mathcal{A}_r$ but allow some affordable inaccuracy of utility. As compensation, we kill SIEVEPAIT instances more conservatively and slightly more instances will be kept to preserve the solution quality.

We still restore $\mathcal{A}_r$ from its observed successor in $\mathbf{l}_t$; however, this will incur some inaccuracy. To avoid killing SIEVEPAIT instances that are actually not redundant, we assign each instance $\mathcal{A}_r$ an amount of *uncertainty*, denoted by β_r, as compensation for inaccurately restore. In other words, the true utility may be as large as $\mathcal{A}_r + \beta_r$. We thus relax the redundancy condition to $\mathcal{A}_j \geq (1-\epsilon)(\mathcal{A}_i + \beta_i)$. Comparing with the original redundancy condition, slightly more instances will be kept.

Uncertainty score β_r is related to the inaccuracy of the instance $\mathcal{A}_r$. Notice that if an instance indexed by r is removed in interval (i, j), we can approximate the interval's uncertainty by $\mathcal{A}_i - \mathcal{A}_j \leq \epsilon\mathcal{A}_i$. Hence, we set $\beta_r = \epsilon(\mathcal{A}_i + \beta_i)$. Therefore, we only need to record each interval's uncertainty when reducing redundancy, and the final implementation of HISTIT-RED is given in Algorithm 4.

VII. EXPERIMENTS

In this section, we perform experiments on several public available real-world datasets to demonstrate the effectiveness of our algorithms.

A. Datasets

• **Brightkite and Gowalla** [40]. Brightkite and Gowalla are two location based online social networks (LBSNs) where users can check in places. To fit into our framework, we can consider a network containing two types of nodes, i.e., user nodes and place nodes. A place node can influence a user node if the place attracts the user to check in. Thus each check-in record reflects the place's influence on the user, and if many users check in a specific place, then the place is influential. We use these two LBSN datasets to study the problem of tracking influential places over check-in record streams.

Algorithm 4 HISTIT-RED

$\mathbf{l} \leftarrow \emptyset$;
foreach *social action* a **do**
  Obtain segments $[l_i(a), r_i(a)], 1 \le i \le n$;
  foreach *segment* $[l, r]$ **do** `Process`(l, r);
  $S_t \leftarrow$ output of $\mathcal{A}_{l_1}$;
  if $l_1 = 0$ **then** Kill $\mathcal{A}_{l_1}$, and $\mathbf{l} \leftarrow \mathbf{l} \backslash \{l_1\}$;
  for $l \in \mathbf{l}$ **do** $\mathcal{A}_{l-1} \leftarrow \mathcal{A}_l$, $l \leftarrow l - 1$;
  `ReduceRedundancy()`;
Procedure `Process`(l, r)
  if $r \notin \mathbf{l}$ **then**
    if r *has no successor* **then** create a new $\mathcal{A}_r$;
    else
      Let i and j denote r's predecessor (or $i = 0$ if not exists) and successor, respectively;
      $\mathcal{A}_r \leftarrow$ a copy of $\mathcal{A}_j$;
      Find $a, b \in \mathbf{l}$ s.t. $(a, b) \supseteq (i, j)$, $\beta_r \leftarrow \beta_{ab}$;
  Feed the segment's I-set to $\mathcal{A}_i$, $i \in [l, r]$;
Procedure `ReduceRedundancy()`
  foreach $i \in \mathbf{l}$ **do**
    Find the largest $j > i$ s.t. $\mathcal{A}_j \ge (1 - \epsilon)(\mathcal{A}_i + \beta_i)$;
    For each $l \in (i, j)$, kill $\mathcal{A}_l$ and remove l from $\mathbf{l}$;
    $\beta_{ij} \leftarrow \epsilon(\mathcal{A}_i + \beta_i)$;

TABLE II
SUMMARY OF DATASETS

dataset	#nodes	#activities	date range
Brightkite [40]	773K/51K	4.7M	Mar/2008 ∼ Oct/2010
Gowalla [40]	1.3M/107K	6.4M	Feb/2009 ∼ Oct/2010
Reddit [41]	2.4M	64.6M	Jan/1 ∼ 31/2016
Twitter [42]	305K	563K	Jul/1 ∼ 7/2012
Weibo [43]	127K	1.5M	Jan/2012 ∼ Dec/2012

• **Reddit** [41]. Reddit is a popular online forum where users can post subjects and comment to other users' posts. If a user v replies to another user u's posts (either subjects or comments), it then reflects user u's influence on user v. Thus if a user attracted many comments, then the user is influential. We use the Reddit data to study the problem of tracking influential users over the commenting streams in Reddit.

• **Twitter** [42]. The Higgs Twitter dataset is built after monitoring the spreading of rumors about Higgs boson on Twitter in July 2012. In Twitter, a user can retweet, reply, and mention another user's tweets, thus reflecting one user's influence to another. Therefore, the Twitter dataset is suitable to study the problem of tracking influential users over the tweets streams in Twitter.

• **Weibo** [43]. Weibo is a popular microblogging website in China. Similar to Twitter, a Weibo user can retweet, reply, and mention another user's tweets, reflecting one user's influence to another. We collected a small fraction of Weibo data using the breath-first-search method starting from some randomly selected seed users in 2012. Similar to Twitter, we use the Weibo dataset to study the problem of tracking influential users over the tweets streams in Weibo.

A brief summary of these datasets is shown in Table II. For Brightkite and Gowalla, we show the number of place nodes and the number of user nodes in each dataset.

B. Baselines

We will consider the following methods as our baselines.

• **Greedy** [29]. A straightforward method to solve the influencer tracking problem is to re-run the Greedy algorithm on $\mathcal{I}_t$ at each time t. The Greedy algorithm starts with an empty set $S = \emptyset$ and iteratively, in each time step, adds a user s who maximizes the utility gain. Greedy stops once it has selected k users, or the utility gain becomes zero. To further improve its efficiency, we apply the lazy evaluation trick [44], which is commonly used to accelerate Greedy. Greedy can find solutions with the best quality, i.e., $1-1/e$ to the optimal, thus Greedy will serve as an upper bound of solution quality.

• **DIM** [23] is a fully-dynamic index data structure for influence analysis on evolving networks with edge additions and deletions. DIM has a parameter β related to the threshold of subgraph weight and also the complexity of the index. We set $\beta = 32$ as suggested in [23].

• **IMM** [14] is an index-based IM method using martingales, and it is designed for handling static graphs. IMM has two parameters l and ϵ, and we set $l = 1$ and $\epsilon = 0.3$.

• **TIM+** [12] is an index-based IM method using the two-phase strategy for static graphs. IMM also has two parameters l and ϵ, and we set $l = 1$ and $\epsilon = 0.3$.

• **Random**. We can randomly pick a set of k users at each time t. The Random method will serve as a lower bound of solution quality.

C. Settings and Evaluation Metrics

The decay function has the form $h(x) = p_0 e^{-\lambda x}$ where we fix $p_0 = 1$ and vary λ. Parameter ϵ and budget k also vary. Sample size n will be determined empirically.

We use the following metrics in the experiments.

· The **Normalized Rooted Mean Squared Error** (NRMSE) is used to evaluate the estimation accuracy of estimator $\hat{\theta}$ for approximating the true value θ. NRMSE is defined by $\text{NRMSE}(\hat{\theta}) \triangleq \sqrt{\mathbb{E}[(\hat{\theta} - \theta)^2]}/\theta$.

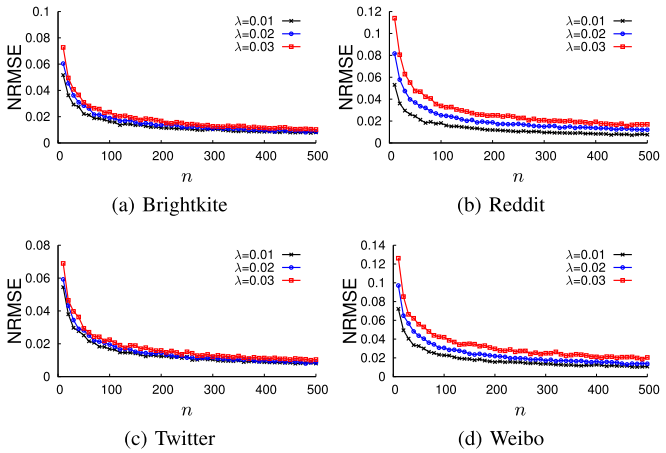
· The **Number of Oracle Calls** is used to evaluate the computational efficiency. Because oracle calls are the most expensive operations in an algorithm, and this metric is independent of the algorithm implementation and experimental hardware.

• **Throughput** is also used as an efficiency metric for handling data streams by a streaming algorithm.

• **Utility** is used to evaluate the quality of solutions found by an algorithm, defined by Eq. 3. A larger utility $F_t(S)$ means that a better solution S is found by the algorithm.

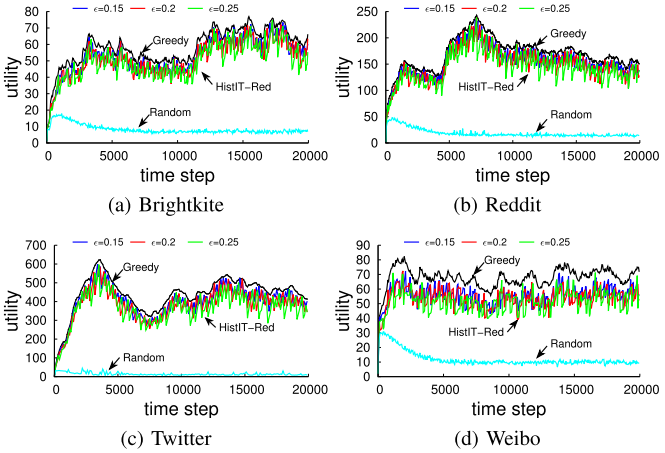
D. Results

• **Sample Size.** We first determine the required sample size n in the Monte-Carlo framework in order to guarantee that $F_t(S)$ is a good estimate of $\mathbb{E}[f(\sigma(S; \mathcal{S}_t, h))]$. To this end, we randomly pick $k = 10$ nodes as node set S, and then compute $F_t(S)$ using different sample size, as well as the true value $\mathbb{E}[f(\sigma(S; \mathcal{S}_t, h))]$, on different datasets. Here, we use small k for the convenience of computing the exact value of $\mathbb{E}[f(\sigma(S; \mathcal{S}_t, h))]$. We then use the NRMSE to evaluate the accuracy of estimate $F_t(S)$. We vary n from 10 to 500, and for each n, we vary λ in range $\{0.01, 0.02, 0.03\}$ and repeat to perform the data stream sampling for $1,000$ times to calculate NRMSE. The results are shown in Fig. 9.



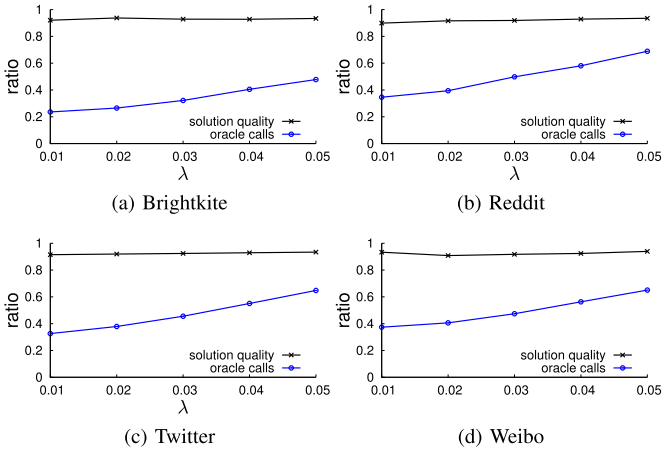
(a) Brightkite (b) Reddit (c) Twitter (d) Weibo

Fig. 9. Estimation accuracy using different sample size. The result on Gowalla is similar to the result on Brightkite.



(a) Brightkite (b) Reddit (c) Twitter (d) Weibo

Fig. 11. Solution quality over time. The result on Gowalla is similar to the result on Brightkite.



(a) Brightkite (b) Reddit (c) Twitter (d) Weibo

Fig. 10. BASICIT vs. HISTIT-RED. The result on Gowalla is similar to the result on Brightkite.

We observe that as sample size n increases, the NRMSE decreases for each λ, as expected. As n increases from 10 to 100, NRMSE drops significantly; when n further increases, NRMSE drops slowly. When $n = 50$, NRMSE is less than 0.05 on all of these five datasets. We therefore fix $n = 50$ in the following experiments.

• **BASICIT vs. HISTIT-RED.** To better understand the improvement of HISTIT-RED upon BASICIT, in the second experiment, we compare the solution quality and computational efficiency between HISTIT-RED and BASICIT.

We run the two algorithms for $1,000$ time steps, and vary λ in range $\{0.01, 0.02, 0.03, 0.04, 0.05\}$, in order to expose the weakness of BASICIT when processing slowly decaying streams. We set $\epsilon = 0.2$, $k = 10$, and $L = 100$, i.e., the maximum lifespan in BASICIT is 100. We calculate two ratios: (1) the *solution quality ratio*, i.e., the quality of solution obtained by HISTIT-RED against the quality of solution obtained by BASICIT, and (2) the *number of oracle calls ratio*, i.e., the number of oracles of HISTIT-RED against the number of oracles calls of BASICIT. Both results are averaged after running $1,000$ time steps, as shown in Fig. 10.

We observe the following results on these datasets. First, the solution quality of HISTIT-RED is slightly lower than that of BASICIT, due to the approximation nature of HISTIT-RED. However, they indeed achieve similar solution quality as the solution quality ratio is larger than 0.9. This indicates that HISTIT-RED can find similar quality solutions as BASICIT. Second, the number of oracle calls of HISTIT-RED is much smaller than that of BASICIT, and the superiority is more significant for smaller λ. This is due to the reason that HISTIT-RED runs smaller number of SIEVEPAIT instances than BASICIT, thus HISTIT-RED requires fewer oracle calls than BASICIT, and for slowly decaying streams (i.e., small λ), BASICIT needs to maintain even more SIEVEPAIT instances, resulting more oracle calls than HISTIT-RED. The experiment implies that HISTIT-RED and BASICIT can find similar quality solutions but HISTIT-RED is more efficient than BASICIT.

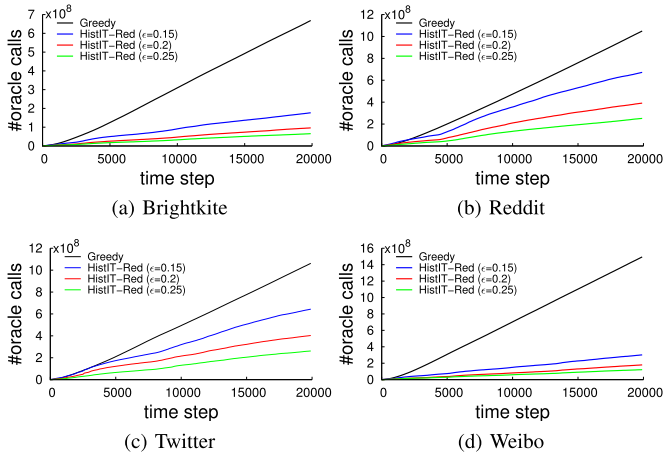
• **Solution Quality.** To understand the solution quality of HISTIT-RED, next we compare the solution quality of HISTIT-RED with Greedy and Random, which serve as the upper and lower bound of solution quality, respectively.

We run HISTIT-RED with fixed $\lambda = 0.001$ and varying ϵ for $20,000$ time steps on each dataset to maintain $k = 20$ nodes at each time step. We show the solution quality of these k nodes over time in Fig. 11.

As expected, the Greedy algorithm always achieves the highest solution quality while the Random algorithm always achieves the lowest. In general, the solution quality achieved by HISTIT-RED is very close to Greedy, and is much better than Random. From the results, ϵ's effect seems to be unclear, and the curves overlap with each other. To further investigate how ϵ affects the solution quality, we show the averaged solution quality ratio with respect to Greedy using different ϵ on each dataset in Fig. 13(a). It is clear to see that when ϵ is larger, the solution quality drops. As larger ϵ means that fewer SIEVEPAIT instances are maintained, thereby resulting in poorer solution quality.

• **Computational Efficiency.** Next we compare the computational efficiency of HISTIT-RED with Greedy in terms of the required number of oracle calls. Similar to the previous experiment, we run HISTIT-RED with fixed $\lambda = 0.001$ and varying ϵ for $20,000$ time steps on each dataset to maintain $k = 20$ nodes at each time step. The accumulated number of oracle calls over time on each dataset is shown in Fig. 12.

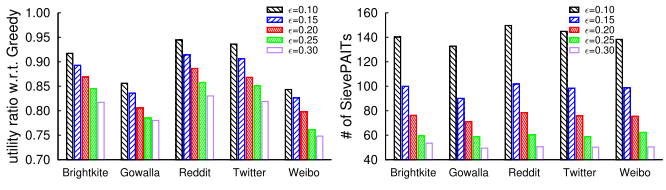
It is clear to see that HISTIT-RED requires much less oracle calls than Greedy, and Greedy is only efficient at the very



(a) Brightkite (b) Reddit

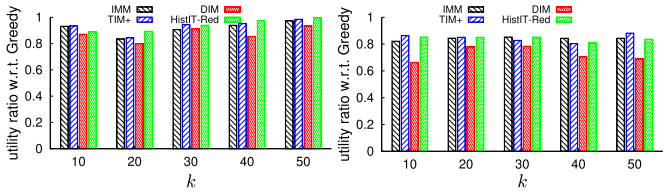
(c) Twitter (d) Weibo

Fig. 12. Computational efficiency. The result on Gowalla is similar to the result on Brightkite.



(a) utility value ratio (b) number of running instances

Fig. 13. Trade-off effect of parameter ϵ.



(a) Twitter (b) Weibo

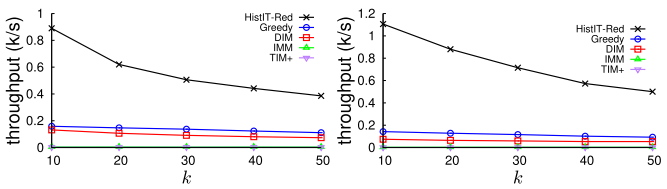
Fig. 14. Solution quality comparision with baselines.

beginning of each stream (when the number of nodes is small) but the number of oracle calls increases quickly over time. For HISTIT-RED, when ϵ increases, the number of oracle calls further drops. Hence, HISTIT-RED is more efficient when ϵ is small due to the fact that a few SIEVEPAIT instances are maintained, as shown in Fig. 13(b).

In combination with the results in Figs. 11 and 13(a), we thus conclude that ϵ has a trade-off effect in HISTIT-RED, i.e., smaller ϵ could improve the solution quality, but harm the computational efficiency.

• **Comparing with Other Baselines.** Next, we compare the solution quality and computational efficiency of HISTIT-RED with other baselines. Here we fix $\lambda = 0.001$ and $\epsilon = 0.2$ for HISTIT-RED. The parameters for each baseline have been described in Section VII-B. We run each method for $1,000$ time steps with varying budget k, and show the averaged results in Figs. 14 and 15.

In Fig. 14, we depict the averaged utility ratio with respect to Greedy for different methods on Twitter and Weibo, respectively. In general, these methods are all able to find similar quality solutions, and they are close to the solution quality of Greedy. In Fig. 15, we depict the throughput of different methods when process the Twitter and Weibo streams. Because Greedy, DIM, IMM, and TIM+ are all not designed for



(a) Twitter (b) Weibo

Fig. 15. Throughput comparision with baselines.

processing streaming data, their throughput is significantly lower than HISTIT-RED. This demonstrates the superiority of HISTIT-RED for processing streams.

VIII. RELATED WORK

Influence Maximization (IM) is the key behind many real-world applications such as viral marketing [45], network monitoring [43], and incentive mechanisms design [46]. Most existing works on IM mainly focus on static networks, e.g., [10], [11], [12], [13], [14], [15], [16], [17], [19], [47], [48], and [49]. While our focus in this work is more related to IM on dynamic networks. Heuristic methods such as [21] and [22] have been proposed to solve IM on dynamic networks. However, these methods do not have theoretical guarantees on the solution quality. Online influence maximization (OIM) methods [50], [51] can provide solution quality guarantees. However, OIM still assumes that the influence is static but allows influence probabilities to be unknown in advance and learned in an online manner.

Song et al. [25] extend the interchange greedy algorithm [29] and propose an $(1/2 - \epsilon)$-approximate algorithm to solve IM over continuously evolving dynamic networks under the assumption that the network changes smoothly so that a few interchange of nodes in the previous solution will obtain a solution at present. However, if the network changes significantly, the interchange greedy degrades to re-computation which is inefficient.

The reverse-reachable sets method [10] is the state-of-the-art for solving IM on static networks. Ohsaka et al. [23] and Yang et al. [24] extend this algorithm to an updatable index structure for faster computing the influence of nodes in dynamic networks. Yang et al. [24] study the problem of tracking influential individuals in dynamic networks, which is a different problem. Ohsaka et al. [23] propose DIM which is a fully-dynamic IM algorithm on networks with edge additions and deletions. Note that these methods still need the pairwise influence probabilities as given inputs and nodes influence is estimated under the IC or LT model. While our approach is a data-driven approach without any assumption on influence spreading model.

Some variants of IM such as topic-specific influencers query [28], [52], finding temporal influencers [53], online learning based approaches [50], [51], and adaptive influence maximization [54] are also related to dynamic networks or dynamic diffusion processes. However, our streaming problem setting is quite different from theirs.

Our work is most related to the researches on streaming submodular optimization (SSO) [55] because the utility function in our influencer tracking problem satisfies submodularity.

In the literature, SSO is mainly studied for two types of data stream models, i.e., the insertion-only stream model [33] and the sliding-window stream model [34], [35], both ensure constant approximation ratios. However, these two stream models actually represent two extremes and have limitations [56]. Recently, SSO algorithms for other data stream models are also studied such as the inhomogeneous decay stream model [36] and temporal biased stream model [57]. Nevertheless, how to adapt these SSO techniques to handle the social activity data under the PDSAS model in the form of streaming graphs remains a challenging problem.

For streams related to online social networks, Wang et al. [27] propose a streaming algorithm to solve IM over sliding-window dynamic social streams and achieve a constant approximation ratio. Zhao et al. [58] propose a streaming algorithm to solve IM over time-decaying dynamic interaction networks where each edge is associated with a fixed lifetime. Our work is different from these existing works on the underlying stream model, and we actually generalize these works by allowing edges in the graph to decay in a probabilistic manner.

IX. Conclusion

In this work, we studied the influencer tracking problem under the PDSAS model to address the dynamic influence challenge. The proposed PDSAS model has the advantage of preserving solution freshness and continuity at the same time. We propose a family of streaming optimization algorithms. SievePAIT can identify influencers from probabilistic addition-only social activity streams efficiently. BasicIT leverages SievePAIT as a building block to identify influencers over general probabilistic-decaying social activity streams. HistIT significantly improves the efficiency of BasicIT, and finally HistIT-Red is a practical implementation of HistIT. We theoretically and empirically show that our approach can find near-optimal solutions with both time and space efficiency.

Proof of Lemma 2

Proof: Let $X_i \triangleq \frac{1}{n} f(\sigma(S;\mathcal{S}_t^{(i)}))$ for $i = 1, \ldots, n$. Then, $X_i \in [0, \frac{1}{n} f(\sigma(S;\mathcal{S}_t))]$ and $F_t(S) = \sum_{i=1}^{n} X_i$. By Hoeffding's inequality, we have

$$\begin{aligned} &P\left(\left|F_t(S) - \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]\right| \geq \epsilon \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]\right) \\ &\leq 2\exp\left(-\frac{2n\epsilon^2 \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]^2}{f(\sigma(S;\mathcal{S}_t))^2}\right) \\ &= 2\exp(-2n\epsilon^2\rho^2) \end{aligned}$$

where $\rho \triangleq \mathbb{E}[f(\sigma(S;\mathcal{S}_t,h))]/f(\sigma(S;\mathcal{S}_t))$. If we expect the above probability is no larger than δ, then we require $n \geq \frac{\ln(2/\delta)}{2\epsilon^2\rho^2}$. □

Proof of Lemma 3

Proof: We first prove the monotonicity of $F_t(S)$. For any sets $S \subseteq T \subseteq V$, by the definition of influence spread, we have $\sigma(S;\mathcal{S}_t^{(i)}) \subseteq \sigma(T;\mathcal{S}_t^{(i)})$. It follows that

$$F_t(S) = \frac{1}{n}\sum_{i=1}^{n} f(\sigma(S;\mathcal{S}_t^{(i)})) \leq \frac{1}{n}\sum_{i=1}^{n} f(\sigma(T;\mathcal{S}_t^{(i)})) = F_t(T)$$

where the inequality holds due to the monotonicity of f.

Next we prove the submodularity of $F_t(S)$, i.e., for any sets $S \subseteq T \subseteq V$ and user $u \in V \backslash T$, we want to show

$$F_t(S \cup \{u\}) - F_t(S) \geq F_t(T \cup \{u\}) - F_t(T).$$

It follows that

$$\begin{aligned} &F_t(S \cup \{u\}) - F_t(S) \\ &= \frac{1}{n}\sum_{i=1}^{n}[f(\sigma(S \cup \{u\};\mathcal{S}_t^{(i)})) - f(\sigma(S;\mathcal{S}_t^{(i)}))] \\ &= \frac{1}{n}\sum_{i=1}^{n}[f(\sigma(S;\mathcal{S}_t^{(i)}) \cup \sigma(u;\mathcal{S}_t^{(i)})) - f(\sigma(S;\mathcal{S}_t^{(i)}))]. \end{aligned}$$

Because f is a submodular function, hence

$$\begin{aligned} &f(\sigma(S;\mathcal{S}_t^{(i)}) \cup \sigma(u;\mathcal{S}_t^{(i)})) - f(\sigma(S;\mathcal{S}_t^{(i)})) \\ &\quad \geq f(\sigma(T;\mathcal{S}_t^{(i)}) \cup \sigma(u;\mathcal{S}_t^{(i)})) - f(\sigma(T;\mathcal{S}_t^{(i)})). \end{aligned}$$

It follows that

$$\begin{aligned} &F_t(S \cup \{u\}) - F_t(S) \\ &\geq \frac{1}{n}\sum_{i=1}^{n}[f(\sigma(T;\mathcal{S}_t^{(i)}) \cup \sigma(u;\mathcal{S}_t^{(i)})) - f(\sigma(T;\mathcal{S}_t^{(i)}))] \\ &= \frac{1}{n}\sum_{i=1}^{n}[f(\sigma(T \cup \{u\};\mathcal{S}_t^{(i)})) - f(\sigma(T;\mathcal{S}_t^{(i)}))] \\ &= F_t(T \cup \{u\}) - F_t(T). \end{aligned}$$

We thus conclude that F_t is a monotone submodular function. □

Proof of Lemma 4

Proof: In incremental sampling, a social activity $a \in \mathcal{S}_{t-1}^{(i)}$ is included in $\mathcal{S}_t^{(i)}$ with probability $e^{-\lambda}$. This random process can be modeled as a Markov chain consisting of two states 1 and 0, where state 1 represents that the social activity is included in $\mathcal{S}_t^{(i)}$, and state 0 denotes not. The transition matrix of this Markov chain is $P = \left[\begin{smallmatrix} 1 & 0 \\ 1-e^{-\lambda} & e^{-\lambda} \end{smallmatrix}\right]$.

Consider a social activity a arrived at time t_a. its state can be represented by $\pi_{t_a} = [1-p_0, p_0]$, i.e., with probability p_0, a is included in a sample. Then at time t, we can obtain that $\pi_t = \pi_{t_a} P^{t-t_a} = [1 - p_0 e^{-\lambda(t-t_a)}, p_0 e^{-\lambda(t-t_a)}]$. It means that, at time t, the element is included in $\mathcal{S}_t^{(i)}$ with probability $p_0 e^{-\lambda(t-t_a)} = p_t(a)$. □

Proof of Theorem 1

Proof: Let $S_t^* = \arg\max_{S \subseteq V \wedge |S| \leq k} F_t(S)$, $\mathrm{OPT}_t \triangleq F_t(S_t^*)$, and $\theta^* = \mathrm{OPT}_t/2k$. We further define $\Delta_t \triangleq \max_{u \in V} F_t(\{u\})$. It is easy to observe that $\max\{\Delta_t, \mathrm{LB}\} \leq \mathrm{OPT}_t \leq k\Delta_t$ and there is a threshold θ such that $(1-\epsilon)\theta^* \leq \theta < \theta^*$.

Let $S_{\theta,t}$ denote the set of users corresponding to threshold θ maintained in SievePAIT at time t. We partition set $S_{\theta,t}$

into two disjoint subsets $S_{\theta,t-1}$ and $S'_{\theta,t}$ where $S_{\theta,t-1}$ is the maintained users at previous time step $t-1$ and $S'_{\theta,t} \subseteq V_a$ is the set of users newly selected from V_a at time t. Our goal is to show that $F_t(S_{\theta,t}) \geq (1/2-\epsilon)\mathrm{OPT}_t$.

We first inductively show that $F_t(S_{\theta,t}) \geq |S_{\theta,t}|\theta$. Thus if $|S_{\theta,t}| = k$, then $F_t(S_{\theta,t}) \geq k\theta \geq (1-\epsilon)\mathrm{OPT}_t/2$.

At time $t=1$, by definition, $S_{\theta,1} = S'_{\theta,1}$ consists of users such that their utility gains are at least θ. Therefore $F_1(S_{\theta,1}) \geq |S_{\theta,1}|\theta$. Assume $F_{t-1}(S_{\theta,t-1}) \geq |S_{\theta,t-1}|\theta$ at time $t-1$. Let us consider $F_t(S_{\theta,t})$ at time t.

$$\begin{aligned} F_t(S_{\theta,t}) &= F_t(S_{\theta,t-1} \cup S'_{\theta,t}) \\ &= \underbrace{F_t(S_{\theta,t-1} \cup S'_{\theta,t}) - F_t(S_{\theta,t-1})}_{\text{first part}} + \underbrace{F_t(S_{\theta,t-1})}_{\text{second part}} \end{aligned}$$

The first part corresponds to the gain of newly selected users $S'_{\theta,t}$ at time t. Because each of the newly selected user has utility gain at least θ by the selection rule of SIEVEPAIT, the first part is at least $|S'_{\theta,t}|\theta$.

For the second part, because a user influence spread cannot decrease in PAIT, we have $F_t(S_{\theta,t-1}) \geq F_{t-1}(S_{\theta,t-1})$. Then by our induction assumption, it follows that the second part is at least $|S_{\theta,t-1}|\theta$.

We thus obtain $F_t(S_{\theta,t}) \geq (|S'_{\theta,t}| + |S_{\theta,t-1}|)\theta = |S_{\theta,t}|\theta$. Therefore, $F_t(S_{\theta,t}) \geq (1-\epsilon)/2 \cdot \mathrm{OPT}_t$ when $|S_{\theta,t}| = k$.

When $|S_{\theta,t}| < k$, we bound the gap between OPT_t and $F_t(S_{\theta,t})$. Using the submodularity of F_t, we have

$$\begin{aligned} \mathrm{OPT}_t - F_t(S_{\theta,t}) &\leq \sum_{u \in S^*_t \setminus S_{\theta,t}} \Delta(u|S_{\theta,t}) \\ &= \sum_{u \in S^*_{t,1} \setminus S'_{\theta,t}} \Delta(u|S_{\theta,t}) + \sum_{v \in S^*_{t,2} \setminus S_{\theta,t-1}} \Delta(v|S_{\theta,t}) \end{aligned}$$

where $S^*_{t,1} \triangleq S^*_t \cap V_a$ is the set of optimal users chosen at time t, and $S^*_{t,2} \triangleq S^*_t \setminus S^*_{t,1}$ is the set of reminder optimal users. For the first part, because these users are not chosen at time t, therefore $\Delta(u|S_{\theta,t}) < \theta$. For the second part, because $S_{\theta,t-1} \subseteq S_{\theta,t}$, then $\Delta(v|S_{\theta,t}) \leq \Delta(v|S_{\theta,t-1})$ due to submodularity. By definition,

$$\begin{aligned} \Delta(v|S_{\theta,t-1}) &= F_t(\{v\} \cup S_{\theta,t-1}) - F_t(S_{\theta,t-1}) \\ &\leq F_{t-1}(\{v\} \cup S_{\theta,t-1}) - F_{t-1}(S_{\theta,t-1}) \\ &\leq \theta. \end{aligned}$$

The first inequality holds due to the fact that the influence spread of $v \notin V_a$ does not increase at t. For the second inequality, we can continue the reasoning as the first inequality until some time $t' < t$ when v's influence spread changed. Because v is not chosen at time t', its utility gain must be less than θ. Therefore, it follows that

$$\mathrm{OPT}_t - F_t(S_{\theta,t}) \leq k\theta \leq k(1+\epsilon)\frac{\mathrm{OPT}_t}{2k} = \frac{1+\epsilon}{2}\mathrm{OPT}_t,$$

which implies $F_t(S_{\theta,t}) \geq (1-\epsilon)\mathrm{OPT}_t/2$.

Because SIEVEPAIT outputs a set of users whose value is at least $F_t(S_{\theta,t})$, we thus conclude that the approximation ratio of SIEVEPAIT is $(1-\epsilon)/2 \geq (1/2-\epsilon)$. □

PROOF OF THEOREM 2

Proof: Notice that the threshold set Θ contains at most $\log_{1+\epsilon} 2k = O(\epsilon^{-1}\log k)$ thresholds.

• **Update complexity**. For each user, we need to calculate the utility gain $|\Theta|$ times (with respect to each S_θ) resulting in $O(n\epsilon^{-1}\log k)$ oracle calls of function f. Because each social activity cause γ users to be added in the user stream, hence the update time complexity is $O(\gamma n\epsilon^{-1}\log k)$.

• **Space complexity**. SIEVEPAIT needs to maintain sets $\{S_\theta \colon \theta \in \Theta\}$ in memory. According to the analysis in SIEVESTREAMING++, for thresholds $\theta \leq \mathrm{LB}/k$, $|S_\theta|$ is trivial upper bounded by k and there are $\log_{1+\epsilon} 2 = O(\epsilon^{-1})$ such thresholds; for thresholds $\theta > \mathrm{LB}/k$, $|S_\theta|$ decreases as θ increases by a factor of $(1+\epsilon)$. Thus the space complexity is $O(k\epsilon^{-1})$. □

PROOF OF LEMMA 5

Proof: If no other instance is created between $\mathcal{A}_{l_i}$ and $\mathcal{A}_{l_{i+1}}$ after they were created, then $\mathcal{A}_{l_i}$ and $\mathcal{A}_{l_{i+1}}$ will be consecutive at time t. This happens when there is no segment $[l,r]$ with r falling between l_i and l_{i+1}, which is Case 1.

Otherwise, $\mathcal{A}_{l_i}$ and $\mathcal{A}_{l_{i+1}}$ may not be consecutive at the time they were both created but became consecutive due to the removal of redundant instances between them. Suppose the redundant instances were removed at the most recent time $t' < t$. The `ReduceRedundancy` procedure then ensures that $\mathcal{A}'_{l_{i+1}} \geq (1-\epsilon)\mathcal{A}'_{l_i}$ at time t'. From time t' to t, it is also impossible to have new indexes being created between these two instances. Otherwise we will meet a contradiction: either redundant instances were created again thus t' is not the most recent time as claimed, or non-redundant indexes were created thus $\mathcal{A}_{l_i}$ and $\mathcal{A}_{l_{i+1}}$ cannot be consecutive at time t. We thus get Case 2. □

PROOF OF THEOREM 5

Proof: If $l_1 = 0$ at time t, i.e., instance $\mathcal{A}_0$ exists. Because $\mathcal{A}_0$'s input is exactly $\mathcal{I}_t$, hence we have

$$\mathcal{A}_{l_1} = \mathcal{A}_0 \geq \frac{1-\epsilon}{2}\mathrm{OPT}_t.$$

Otherwise $l_1 > 0$ at time t. In this case, instance $\mathcal{A}_{l_1}$ actually processed a family of I-sets that is different from $\mathcal{I}_t$, and hence incurs a loss on solution quality. Below, we show that this loss can be bounded.

Note that indexes in $\mathbf{l}_t$ will gradually shift to the left and then expire. Let l_0 be the most recent expired predecessor of l_1 at time t. Because l_0 and l_1 were two consecutive indexes (at the time l_0 did not expire), then by Lemma 5, we shall have two cases.

• **Case 1 holds.** Then, no social activity has a segment with the right end lying between $\mathcal{A}'_{l_0}$ and $\mathcal{A}'_{l_1}$ since they were created. In other words, instance $\mathcal{A}_{l_1}$ has the same inputs as $\mathcal{A}_0$ because their input I-sets always belong to the same segment. Hence, $\mathcal{A}_{l_1}$ and $\mathcal{A}_0$ have the same output at time t. So we have

$$\mathcal{A}_{l_1} = \mathcal{A}_0 \geq \frac{1-\epsilon}{2}\mathrm{OPT}_t.$$

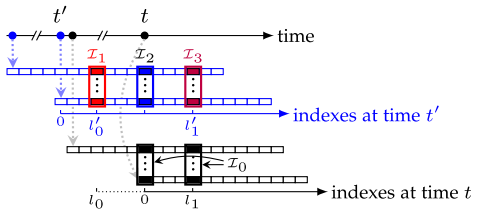


Fig. 16. Indices at time t', t, and t.

• **Case 2 holds.** Then, there exists time $t' < t$ such that

$$\mathcal{A}'_{l_1} \geq (1-\epsilon)\mathcal{A}'_{l_0}, \tag{6}$$

and from t' to t, no instance is created between $\mathcal{A}'_{l_0}$ and $\mathcal{A}'_{l_1}$.

Let us clarify the relation of the inputs of three SIEVEPAIT instances $\mathcal{A}_{l_0}$, $\mathcal{A}_0$, and $\mathcal{A}_{l_1}$. Note that at time t, only $\mathcal{A}_{l_1}$ is practically maintained, while the other two actually do not exist. At time t', let $\mathcal{I}_1$, $\mathcal{I}_2$, and $\mathcal{I}_3$ denote the processed I-sets of $\mathcal{A}'_{l_0}$, $\mathcal{A}'_0$, and $\mathcal{A}'_{l_1}$, respectively, as illustrated in Fig. 16. From t' to t, as claimed in Case 2 of Lemma 5, the I-sets fed to $\mathcal{A}_0$ and $\mathcal{A}_{l_1}$ always belong to the same segment. Hence, $\mathcal{A}_0$ and $\mathcal{A}_{l_1}$ have the same inputs in time interval $(t', t]$, denoted by $\mathcal{I}_0$, as illustrated in Fig. 16. At time t, instance $\mathcal{A}_0$ has processed I-sets $\mathcal{I}_2||\mathcal{I}_0$, and $\mathcal{A}_{l_1}$ has processed I-sets $\mathcal{I}_3||\mathcal{I}_0$, where $||$ denotes the concatenation of two I-set streams. We want to find out how close the output of $\mathcal{A}_{l_1}$ is to that of $\mathcal{A}_0$.

To clearly point out the inputs of each SIEVEPAIT instance, we let $\mathcal{A}(\mathcal{I})$ denote the output utility of instance $\mathcal{A}$ when the input is I-sets $\mathcal{I}$. Similarly, let $\mathrm{OPT}(\mathcal{I})$ denote the utility of an optimal solution found in $\mathcal{I}$. Let $F(S;\mathcal{I})$ denote the utility of a user set S in I-sets $\mathcal{I}$. Note that I-sets $\mathcal{I}$ are equivalent to n social activity samples, denoted by $\mathcal{I}^{(i)}, i = 1, \ldots, n$. Then, on each sample, we can evaluate the influence scopes of users, hence $F(S;\mathcal{I}) = \frac{1}{n}\sum_{i=1}^{n} f(\sigma(S;\mathcal{I}^{(i)}))$.

At time t, we have

$$\begin{aligned}
\mathcal{A}_{l_1} = \mathcal{A}(\mathcal{I}_3||\mathcal{I}_0) &= \frac{1}{2}\big(\mathcal{A}(\mathcal{I}_3||\mathcal{I}_0) + \mathcal{A}(\mathcal{I}_3||\mathcal{I}_0)\big) \\
&\geq \frac{1}{2}\big(\mathcal{A}(\mathcal{I}_3) + \mathcal{A}(\mathcal{I}_3||\mathcal{I}_0)\big) \\
&\geq \frac{1}{2}\big((1-\epsilon)\mathcal{A}(\mathcal{I}_1) + \mathcal{A}(\mathcal{I}_3||\mathcal{I}_0)\big) \\
&\geq \frac{1-\epsilon}{2}\big(\mathcal{A}(\mathcal{I}_1) + \mathcal{A}(\mathcal{I}_3||\mathcal{I}_0)\big)
\end{aligned}$$

Here, the first inequality holds due to the monotonicity of SIEVEPAIT, i.e., when feeding more I-sets, the output utility of a SIEVEPAIT instance does not decrease. The second inequality holds due to claim (6). Because SIEVEPAIT is $(1-\epsilon)/2$ approximate, hence $\mathcal{A}(\mathcal{I}_1) \geq (1-\epsilon)/2 \cdot \mathrm{OPT}(\mathcal{I}_1)$ and $\mathcal{A}(\mathcal{I}_3||\mathcal{I}_0) \geq (1-\epsilon)/2 \cdot \mathrm{OPT}(\mathcal{I}_3||\mathcal{I}_0)$. It follows that

$$\mathcal{A}_{l_1} \geq \frac{1}{4}(1-\epsilon)^2\big(\mathrm{OPT}(\mathcal{I}_1) + \mathrm{OPT}(\mathcal{I}_3||\mathcal{I}_0)\big).$$

Let S^*_t denote an optimal user set in I-sets $\mathcal{I}_2||\mathcal{I}_0$, i.e., $F(S^*_t;\mathcal{I}_2||\mathcal{I}_0) = \mathrm{OPT}(\mathcal{I}_2||\mathcal{I}_0) = \mathrm{OPT}_t$. Since $\mathrm{OPT}(\mathcal{I}_1)$ is the optimal utility in $\mathcal{I}_1$, we have

$$\mathrm{OPT}(\mathcal{I}_1) \geq F(S^*_t;\mathcal{I}_1).$$

Note that $\mathcal{I}_1$ contains more social activities than $\mathcal{I}_2$, hence

$$F(S^*_t;\mathcal{I}_1) \geq F(S^*_t;\mathcal{I}_2).$$

Since $\mathrm{OPT}(\mathcal{I}_3||\mathcal{I}_0)$ is the optimal utility in $\mathcal{I}_3||\mathcal{I}_0$, hence

$$\mathrm{OPT}(\mathcal{I}_3||\mathcal{I}_0) \geq F(S^*_t;\mathcal{I}_3||\mathcal{I}_0).$$

It follows that

$$\begin{aligned}
&\mathrm{OPT}(\mathcal{I}_1) + \mathrm{OPT}(\mathcal{I}_3||\mathcal{I}_0) \\
&\quad\geq F(S^*_t;\mathcal{I}_2) + F(S^*_t;\mathcal{I}_3||\mathcal{I}_0) \\
&\quad= \frac{1}{n}\sum_{i=1}^{n}\Big[f\big(\sigma(S^*_t;\mathcal{I}_2^{(i)})\big) + f\big(\sigma(S^*_t;\mathcal{I}_3^{(i)}||\mathcal{I}_0^{(i)})\big)\Big] \\
&\quad\geq \frac{1}{n}\sum_{i=1}^{n} f\big(\sigma(S^*_t;\mathcal{I}_2^{(i)}) \cup \sigma(S^*_t;\mathcal{I}_3^{(i)}||\mathcal{I}_0^{(i)})\big) \\
&\quad= \frac{1}{n}\sum_{i=1}^{n} f\big(\sigma(S^*_t;\mathcal{I}_2^{(i)}||\mathcal{I}_0^{(i)})\big) \\
&\quad= F(S^*_t;\mathcal{I}_2||\mathcal{I}_0) = \mathrm{OPT}(\mathcal{I}_2||\mathcal{I}_0) = \mathrm{OPT}_t
\end{aligned}$$

where the second inequality holds due to the submodularity of f. We thus conclude that

$$\mathcal{A}(\mathcal{I}_3||\mathcal{I}_0) \geq \frac{1}{4}(1-\epsilon)^2\mathrm{OPT}_t \geq (\frac{1}{4} - \epsilon)\mathrm{OPT}_t.$$

which completes the proof. □

PROOF OF THEOREM 6

Proof: The reducing redundancy operation ensures that $\mathcal{A}_{l_{i+2}} < (1-\epsilon)\mathcal{A}_{l_i}$ always holds. Because $\mathcal{A}_l \in [\Delta, k\Delta]$ where $\Delta \triangleq \max_u F_t(u)$, then $|\mathbf{l}_t|$ is upper bounded by $O(\log_{(1-\epsilon)^{-1}} k) = O(\epsilon^{-1}\log k)$.

• **Update complexity**. For each user, in the worst case, we need to update $|\mathbf{l}_t|$ SIEVEPAIT instances, and each SIEVEPAIT instance requires $O(\gamma n \epsilon^{-1}\log k)$ oracle calls. Hence the total number of oracle calls for each user is $O(\gamma n \epsilon^{-2}\log^2 k)$.

• **Space complexity**. Because each SIEVEPAIT instance has space complexity $O(k\epsilon^{-1})$, thus maintaining $|\mathbf{l}_t|$ SIEVEPAIT instances requires space $O(k\epsilon^{-2}\log k)$. □

REFERENCES

[1] (May 2022). *Amazon Influencer Program*. [Online]. Available: https://affiliate-program.amazon.com/influencers

[2] (May 2022). *Influencer Marketing: Social Media Influencer Market Stats and Research for 2021*. [Online]. Available: https://www.businessinsider.com/influencer-marketing-report

[3] (May 2022). *To Fight Vaccine Lies, Authorities Recruit an 'Influencer Army'*. [Online]. Available: https://www.nytimes.com/2021/08/01/technology/vaccine-lies-influencer-army.html

[4] (May 2022). *The First TikTok War: How are Influencers in Russia and Ukraine Responding?* [Online]. Available: https://www.theguardian.com/media/2022/feb/26/social-media-influencers-russia-ukraine-tiktok-instagram

[5] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2001, pp. 57–66.

[6] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2003, pp. 137–146.

[7] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *Proc. KES*, 2008, pp. 67–75.

[8] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "Learning influence probabilities in social networks," in *Proc. 3rd ACM Int. Conf. Web Search Data Mining*, Feb. 2010, pp. 241–250.

[9] K. Kutzkov, A. Bifet, F. Bonchi, and A. Gionis, "STRIP: Stream learning of influence probabilities," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2013, pp. 275–283.

[10] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proc. 24th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2014, pp. 946–957.

[11] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2010, pp. 1029–1038.

[12] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2014, pp. 75–86.

[13] B. Lucier, J. Oren, and Y. Singer, "Influence at scale: Distributed computation of complex contagion in networks," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 735–744.

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

[15] W. Chen, T. Lin, Z. Tan, M. Zhao, and X. Zhou, "Robust influence maximization," in *Proc. ACM SIGKDD*, 2016, pp. 795–804.

[16] I. Litou, V. Kalogeraki, and D. Gunopulos, "Influence maximization in a many cascades world," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 911–921.

[17] Y. Lin, W. Chen, and J. C. S. Lui, "Boosting information spread: An algorithmic approach," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 883–894.

[18] X. Ke, A. Khan, and G. Cong, "Finding seeds and relevant tags jointly: For targeted influence maximization in social networks," in *Proc. ACM SIGMOD*, 2018, pp. 1097–1111.

[19] S. Kumar, A. Mallik, A. Khetarpal, and B. S. Panda, "Influence maximization in social networks using graph embedding and graph neural network," *Inf. Sci.*, vol. 607, pp. 1617–1636, Aug. 2022.

[20] S. A. Myers and J. Leskovec, "The bursty dynamics of the Twitter information network," in *Proc. 23rd Int. Conf. World Wide Web*, Apr. 2014, pp. 913–924.

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

[22] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun, "Influence maximization in dynamic social networks," in *Proc. IEEE 13th Int. Conf. Data Mining*, Dec. 2013, pp. 1313–1318.

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

[24] Y. Yang, Z. Wang, J. Pei, and E. Chen, "Tracking influential individuals in dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2615–2628, Nov. 2017.

[25] G. Song, Y. Li, X. Chen, X. He, and J. Tang, "Influential node tracking on dynamic social network: An interchange greedy approach," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 2, pp. 359–372, Feb. 2017.

[26] (May 2022). *Internet Live Stats*. [Online]. Available: https://www.internetlivestats.com/one-second/

[27] Y. Wang, Q. Fan, Y. Li, and K.-L. Tan, "Real-time influence maximization on dynamic social streams," in *Proc. VLDB*, 2017, pp. 805–816.

[28] K. Subbian, C. C. Aggarwal, and J. Srivastava, "Querying and tracking influencers in social streams," in *Proc. 9th ACM Int. Conf. Web Search Data Mining*, Feb. 2016, pp. 493–502.

[29] G. Nemhauser, L. Wolsey, and M. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Program.*, vol. 14, pp. 265–294, Dec. 1978.

[30] A. Krause and D. Golovin, *Submodular Function Maximization*. Cambridge, U.K.: Cambridge Univ. Press, 2014, pp. 71–104.

[31] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Mining maximal cliques from an uncertain graph," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 243–254.

[32] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods* (Springer Texts in Statistics), G. Casella, S. Fienberg, and I. Olkin, Eds., 2nd ed. New York, NY, USA: Springer, 2004.

[33] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: Massive data summarization on the fly," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 671–680.

[34] J. Chen, H. L. Nguyen, and Q. Zhang, "Submodular maximization over sliding windows," 2016, *arXiv:1611.00129*.

[35] A. Epasto, S. Lattanzi, S. Vassilvitskii, and M. Zadimoghaddam, "Submodular optimization over sliding windows," in *Proc. 26th Int. Conf. World Wide Web*, Apr. 2017, pp. 421–430.

[36] J. Zhao, S. Shang, P. Wang, J. C. Lui, and X. Zhang, "Submodular optimization over streams with inhomogeneous decays," in *Proc. AAAI*, 2019, pp. 5861–5868.

[37] E. Kazemi, M. Mitrovic, M. Zadimoghaddam, S. Lattanzi, and A. Karbasi, "Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity," in *Proc. ICML*, 2019, pp. 3311–3320.

[38] V. Braverman and R. Ostrovsky, "Smooth histograms for sliding windows," in *Proc. 48th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Oct. 2007, pp. 283–293.

[39] R. Krauthgamer and D. Reitblat, "Almost-smooth histograms and sliding-window graph algorithms," 2019, *arXiv:1904.07957*.

[40] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2011, pp. 1082–1090.

[41] (May 2022). *Reddit Data Dump*. [Online]. Available: https://files.pushshift.io/reddit/

[42] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi, "The anatomy of a scientific rumor," *Sci. Rep.*, vol. 3, no. 1, pp. 1–9, Oct. 2013.

[43] J. Zhao, J. C. S. Lui, D. Towsley, and X. Guan, "Whom to follow: Efficient followee selection for cascading outbreak detection on online social networks," *Comput. Netw.*, vol. 75, pp. 544–559, Dec. 2014.

[44] M. Minoux, "Accelerated greedy algorithms for maximizing submodular set functions," in *Optimization Techniques*, vol. 7. Berlin, Germany: Springer, 1978, pp. 234–243.

[45] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "A billion-scale approximation algorithm for maximizing benefit in viral marketing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2419–2429, Aug. 2017.

[46] Z. Shi, G. Yang, X. Gong, S. He, and J. Chen, "Quality-aware incentive mechanisms under social influences in data crowdsourcing," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 176–189, Feb. 2022.

[47] S. Feng, G. Cong, A. Khan, X. Li, Y. Liu, and Y. M. Chee, "Inf2vec: Latent representation model for social influence embedding," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 941–952.

[48] K. Zhang, J. Zhou, D. Tao, P. Karras, Q. Li, and H. Xiong, "Geodemographic influence maximization," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 2764–2774.

[49] W. Chen, X. Sun, J. Zhang, and Z. Zhang, "Network inference and influence maximization from samples," in *Proc. ICML*, 2021, pp. 1707–1716.

[50] S. Lei, S. Maniu, L. Mo, R. Cheng, and P. Senellart, "Online influence maximization," in *Proc. ACM SIGKDD*, 2015, pp. 645–654.

[51] S. Li, F. Kong, K. Tang, Q. Li, and W. Chen, "Online influence maximization under linear threshold model," in *Proc. NIPS*, 2020, pp. 1192–1204.

[52] K. Subbian, C. Aggarwal, and J. Srivastava, "Content-centric flow mining for influence analysis in social streams," in *Proc. 22nd ACM Int. Conf. Conf. Inf. Knowl. Manag.*, 2013, pp. 841–846.

[53] S. Huang, Z. Bao, J. S. Culpepper, and B. Zhang, "Finding temporal influential users over evolving social networks," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 398–409.

[54] G. Tong, W. Wu, S. Tang, and D.-Z. Du, "Adaptive influence maximization in dynamic social networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 112–125, Feb. 2017.

[55] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in MapReduce and streaming," in *Proc. ACM SPAA*, 2013, pp. 1–10.

[56] J. Zhao, P. Wang, J. Tao, S. Zhang, and J. C. S. Lui, "Continuously tracking core items in data streams with probabilistic decays," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Apr. 2020, pp. 769–780.

[57] J. Zhao, P. Wang, C. Deng, and J. Tao, "Temporal biased streaming submodular optimization," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2021, pp. 2305–2315.

[58] J. Zhao, S. Shang, P. Wang, J. C. S. Lui, and X. Zhang, "Tracking influential nodes in time-decaying dynamic interaction networks," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1106–1117.