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 In order to keep the solutions up-to-date and forget outdated data in the stream smoothly, we propose a probabilistic-decaying social activity stream (PDSAS) model that enforces each social activity in the stream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

INFLUENCER 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, her 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/TON.2023.3323028

1558-2566 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on June 25,2024 at 07:03:24 UTC from IEEE Xplore. Restrictions apply.
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 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, 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 \(a \triangleq (u, v, t)\) denote a social activity representing that user \(u\) performed a specific activity \(c\) (e.g., purchasing an
When considering the influence of the original social activity that triggers the observed social activity, we evaluate user's influence based on the observed social activities. For example, consider two social activities performed by two users $u$ and $v$ at time $t$ and $t + \delta$, respectively. If $(u, v) \in E$ and $\delta$ 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 $u \xrightarrow{A} v$, if $v$ is reachable from $u$ in $G$ and there exist social activities $(u, c, t)$ and $(v, c, t') \in A$ such that $t' - t \leq \delta$ for some threshold $\delta$.

It is worthy noting that, in the above definition, we do not require $(u, c, t) \in A$ but assume $(u, c, t)$ 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., $\sigma(u; A) \triangleq \{v \mid u \xrightarrow{A} v\}$.

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 $f$ 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, $f$ 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 $v \in V \setminus Y$, 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 $S_t \triangleq \{a \mid t_a \leq t\}$ 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 $a \in S_t$, we let $a$ participate in the analysis with probability $p_t(a) \triangleq h(t - t_a)$, and $h: \mathbb{Z} \mapsto [0, 1]$ 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 $S_t$ 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 $S_t$ 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.,

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

(1)

Here, $\sigma(S;S_t,h)$ 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 utility function in together govern the selection of users in the network. This allows the PDSAS framework to decay inhomogeneously.

**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 $\mathbb{E}[f(\sigma(S;S_t,h))]$ 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., $O(2^{|S_t|})$. To address this challenge, we propose a Monte-Carlo method to fast estimate $\mathbb{E}[f(\sigma(S;S_t,h))]$ 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 $S$ to denote a realization of $S_t$, then $S \subseteq S_t$. Let $S_t \triangleq \{S \mid S \subseteq S_t\}$ 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

$$P(S \in S_t) = \prod_{a \in S_t} p_t(a)^{1(a \in S)}(1 - p_t(a))^{1(a \notin S)}$$

(2)

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

For example, in Fig. 3, we show some realizations of the original stream, and each social activity $a \in S_t$ 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.

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 drawn from an identical distribution can provide an unbiased estimate of the expectation. Specifically, we have the following result.

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

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

(3)

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

Given a sample $S_t^{(i)}$, which is actually a collection of social activities, we can calculate the influence spread $\sigma(s;S_t^{(i)})$ for each user $s \in S$ (recall the example in Fig. 2). Therefore, we can easily obtain $\sigma(S;S_t^{(i)})$ and hence $F_t(S)$. Intuitively, $\mathbb{E}[f(\sigma(S_t;S_t,h))]$ is the average influence of set $S$ on $|S_t|$ 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 $\mathbb{E}[f(\sigma(S_t;S_t,h))]$ almost surely. Therefore, by choosing a sufficiently large sample size $n$, $F_t(S)$ will be a good approximation of $\mathbb{E}[f(\sigma(S_t;S_t,h))]$. 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 $f$. For sample size $n$ can be determined by applying the Hoeffding’s inequality.

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

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

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

In the following discussion, we will assume that sample size $n$ is sufficiently large so that $F_t(S) \approx \mathbb{E}[f(\sigma(S_t;S_t,h))]$. 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 $\{S_t^{(i)}\}_{i=1}^n$ from the original stream $S_t$, given decay function $h(x) = p_0 e^{-\lambda x}$, thereby solving sub-problem (i).

IV. STREAM SAMPLING METHODS

In this section, we design sampling methods to generate stream samples $\{S_t^{(i)}\}_{i=1}^n$ from the original stream $S_t$, 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 $S_t^{(i)}$ from $S_t$ straightforwardly. That is, at each time $t$, we sequentially scan the social activities in the stream; for each social activity $a \in S_t$, we include $a$ in sample $S_t^{(i)}$ 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 $t+1$, 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 $h(x) = p_0e^{-\lambda x}$, we can sample the stream incrementally and obtain $S_t^{(i)}$ from $S_{t-1}^{(i)}$ in a more efficient manner. Specifically, for each social activity $a \in S_{t-1}^{(i)}$, we keep $a$ in $S_t^{(i)}$ with probability $e^{-\lambda}$, and for each newly arrived $a$ at time $t$, we add $a$ in $S_t^{(i)}$ with probability $p_t$. The following lemma states that the obtained sample $S_t^{(i)}$ indeed follows the desired distribution $P(S)$.

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

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

$$P(l_a = l) = \begin{cases} 1 - p_0, & l = 0, \\ p_0e^{-\lambda(l-1)}(1 - e^{-\lambda}), & l = 1, 2, \ldots \end{cases} \tag{4}$$

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

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

Lifespan sampling is a more convenient way to sample $S_t$ 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 $\{S_t^{(i)}\}_{i=1}^n$, the I-set of a social activity $a$ at time $t = t_a + l$ (i.e., $l$ time steps after its arrival) is

$$I_l(a) \triangleq \{ i \mid a \in S_t^{(i)} \} = \{ i \mid t_{ia}^{(i)} > t \}. \tag{5}$$

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 $S_t^{(i)}$ if $t_{ia}^{(i)} > l$. 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. 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 $p_t(a) = p_0 \in [0, 1]$, 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 $I_l(a)$. $I_l(a)$ can be directly obtained by flipping a biased coin (with probability $p_0$ of heads) $n$ times, and $I_l(a) \triangleq \{ i: \text{the } i\text{-th outcome is a head} \}$.

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 $a \in S_t$, 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
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 \( S^{(i)}_t \) 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., \( \sigma(u; S^{(i)}_t) \) 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 \( \{S^{(i)}_t\}_{i=1}^n \), 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

\[
\Delta(u|S) \triangleq F_t(S \cup \{u\}) - F_t(S)
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \left( f(\sigma(S \cup \{u\}; S^{(i)}_t)) - f(\sigma(S; S^{(i)}_t)) \right)
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} \delta^{(i)}(u|S)
\]

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 \( t \)

1. \( \Delta \leftarrow 0 \), \( \theta_{\text{min}} \leftarrow \theta \), and \( \text{LB} \leftarrow 0 \)

2. foreach social activity \( a \) arrived at time \( t \) do

3. \( V_a \leftarrow \) users whose influence spreads change;

4. // Lines 4-7 maintain a set of thresholds

5. \( \Delta \leftarrow \max\{\Delta, \max_{u \in V_a} F_t(\{u\})\} \)

6. \( \theta_{\text{min}} \leftarrow \max\{\text{LB}, \Delta/2k\} \)

7. Discard all sets \( S_0 \) with \( \theta < \theta_{\text{min}} \);

8. \( \Theta \leftarrow \{(1 + \epsilon)^t | (1 + \epsilon)^t \in [\theta_{\text{min}}/(1 + \epsilon), \Delta]\} \)

9. // Lines 8-12 filter users by thresholds

10. foreach \( u \in V_a \) do

11. // if \( |S_0| < k \) and \( \Delta(u|S_0) \geq \theta \) then

12. \( S_{t} \leftarrow S_{t} \cup \{u\}; \)

13. \( \text{LB} \leftarrow \max\{\text{LB}, F_t(S_{t})\} \)

14. // Return \( S_t \) when a query is performed

\( S_t \leftarrow \arg \max F_t(S_{t}) \);

---

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 \( \gamma \) denote the average size of set \( V_a \) for all \( a \). Then SIEVEPAIT has the following complexity.

**Theorem 2:** SIEVEPAIT requires \( O(\gamma \epsilon^{-1} \log k) \) 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 \( I(a) \ | \ a \in S_t \), and output is an \((1/2 - \epsilon)\)-approximate solution of the PAIT problem. We use \( A \) 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.
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 $I_1(a) = \emptyset$ 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 $t$, $a_2$ 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 continues, and when an I-set becomes empty, the corresponding social activity no longer exists.

Let $I_t = \{l(a) \neq \emptyset \mid a \in S_t \land t_a + l \leq t\}$ be a family of non-empty I-sets at time $t$, formed by currently alive social activities. For example, in Fig. 6(a), $I_t = \{I_2(a_1), I_1(a_2), I_0(a_3)\}$. In fact, at time $t$, we can treat $I_t$ 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 $p_l(a_1)$ and its I-set is $I_2(a_1)$; $a_2$ participates in the analysis with an instantaneous constant probability $p_l(a_2)$ 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 $I_t$ 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 $I_t$ are evolving over time, and we can access each social activity only once. We cannot afford to process $I_t$ again at each time $I_t$ changes. Therefore, we need to find a clever way to process $I_t$ 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 $\{A_l\}_{l=0}^{L-1}$, 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 $A_l$ for $l = 0, \ldots, L - 1$.
- **Shifting.** Reset $A_0$ 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 $A_0$ always processed all the I-sets in $I_t$, thus $A_0$’s output will be the solution at time $t$.

The pseudo-code of BASICIT is given in Algorithm 2.

![Diagram](https://via.placeholder.com/150)

**Algorithm 2 BASICIT**

1. Initialize $L$ SievePAIT instances $\{A_l\}_{l=0}^{L-1}$
2. foreach social action $a$ do
   3. Obtain I-sets $I_l(a), l = 0, \ldots, L - 1$
   4. for $l = 0, \ldots, L - 1$ do Feed $A_l$ with $I_l(a)$
   5. $S_t \leftarrow$ output of $A_0$
   6. for $l = 1, \ldots, L - 1$ do $A_{l-1} \leftarrow A_l$
   7. Reset $A_0$ and $A_{L-1} \leftarrow A_0$

In Fig. 7, we use an example to explain the execution of BASICIT. Assume $L = 3$, and we maintain three SievePAIT instances, namely $A$, $A'$, and $A''$. When $a_3$ arrives, we obtain its I-sets and feed them to the three instances, respectively. Then $A$ 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, $A''$ has processed I-sets $I_2(a_3), I_1(a_2)$, and $I_0(a_3)$ which are exactly the I-sets in $I_t$ (see Fig. 6(a)). Hence, $A''$’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 $O(Ln\text{yc}^{-1} \log k)$ 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.
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 \( I_1 = \{ l_1^i, l_2^i, \ldots \} \), where \( l_i^j \in \{0, 1, \ldots \} \) is an 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 \( I_1(a) = I_1(a) \), \( I_2(a) = I_2(a) = I_4(a) \), and \( I_3(a) = I_6(a) \). 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 \( 1 \leq i \leq n \). 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 \( \{ I_i(a) \}_{i \geq 0} \). Because \( I_1(a) \) will be fed to \( A \) and I-sets belong to the same segment are same to each other, hence \( A \) 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 \( I_l = \{ r_l(a) \}_{1 \leq l \leq n} \). For example, in Fig. 8(a), because \( a_1 \) has two segments, we only need to maintain two instances, i.e., \( A_l \) 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 \( I_{l-1} \) denote the set of indexes of maintained instances before processing \( a \). For each segment \([l, r]\) of \( a \), if \( r \not\in I_{l-1} \), we can create \( A_r \) by copying the successor instance \( A_{l'} \) where \( l' = \min\{l \mid l > r \land l \in I_{l-1} \} \); if no such successor exists, we create a new instance as \( A_r \). We add \( r \) in \( I_{l-1} \) and obtain \( I_l \). Finally, for each \( l \in I_{l-1} \), we also need to feed \( A_l \) with the I-set of the segment that \( l \) belongs to.

For example, in Fig. 8(b), \( a_2 \) arrives at time \( t \), and \( I_{l-1} = \{ r_1(a_1), r_2(a_1) \} \). Because \( r_1(a_2) \not\in I_{l-1} \) and \( r_2(a_2) \not\in I_{l-1} \), we hence create two instances \( A_{r_1(a_2)} \) and \( A_{r_2(a_2)} \) by copying the same successor \( A_{r_2(a_1)} \). Then we feed segment one’s I-set to \( A_{r_1(a_1)} \) and \( A_{r_2(a_2)} \), and feed segment two’s I-set to \( A_{r_2(a_2)} \). 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: \( I \leftarrow \emptyset; \)
2: foreach social action \( a \) do
3: \( \text{Obtain segments } [l_i(a), r_i(a)], 1 \leq i \leq n; \)
4: foreach segment \([l, r]\) do \( \text{Process}(l, r); \)
5: \( S_l \leftarrow \text{output of } A_l; \)
6: if \( l_1 = 0 \) then \( \text{Kill } A_{l_1}, \) and \( I \leftarrow I \setminus \{l_1\}; \)
7: for \( l \in I \) do \( A_{l-1} \leftarrow A_l, \) and \( I \leftarrow I \setminus l; \)
8: \( \text{Procedure}(l, r); \)
9: if \( r \not\in I \) then
10: \( \text{if } r \text{ has no successor then create a new } A_r; \)
11: \( \text{else create } A_r \text{ by copying its successor; } \)
12: \( I \leftarrow I \cup \{r\}; \)
13: \( \text{Feed the segment's I-set to } A_i, \) \( i \in [l, r]; \)

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 \( A \) to refer to the instance’s output utility.

Definition 4: Given a small real number \( \epsilon \in (0, 1) \), if \( A_j \geq (1 - \epsilon)A_i \) for \( j > i \), then we say that instances with indexes between \( i \) and \( j \) are \( \epsilon \)-redundant.

That is, since \( A_j \) and \( A_i \) are already close to each other, then instances between them are considered to be redundant.
Thus, we can regularly check the output utilities of SIEVE-PAIT 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 \( A_r \) were killed at some previous step (because \( A_r \) were redundant), then at current time, the maintained successor is actually not the true successor of \( A_r \). Creating \( A_r \) by copying the fake successor may cause issues. We postpone the solution to this issue, but simply assume that there 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, \( l \in I_t \) and \( l+1 \in I_{t-1} \) actually index the same SIEVEPAIT instance, i.e., \( A_l \) at time \( t \) and \( A_{l+1} \) at time \( t-1 \) are the same SIEVEPAIT instance. In general, \( l' \in I_{t'} \) and \( l \in I_t \) index the same instance if \( t' \leq t \) and \( l' = l + t - t' \). If so, we say that \( A_{l'} \) at time \( t' \) is an ancestor of \( A_l \) at time \( t \). To simplify the notations, we will always use \( A_l' \) to refer to an ancestor instance of \( A \). We find that HISTIT-RED maintains a histogram satisfying the following property.

**Lemma 5:** Let \( A_{l_1} \) and \( A_{l_{i+1}} \) 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 \( A_{l_{i+1}}' \geq (1 - \epsilon)A_{l_{i}'} \), 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 \( A_0 \) exists. Because \( A_0 \) processed all the I-sets in \( I_t \) 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 \( A_{l_1} \) still completely processed the I-sets in \( I_t \), 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 \( A_r \), 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 \( A_r \) 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 \( A_r \) 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 \( A_r \) from its observed successor in \( I_1 \); however, this will incur some inaccuracy. To avoid killing SIEVEPAIT instances that are actually not redundant, we assign each instance \( A_r \) an amount of uncertainty, denoted by \( \beta_r \), as compensation for inaccurately restore. In other words, the true utility may be as large as \( A_r + \beta_r \). We thus relax the redundancy condition to \( A_j \geq (1 - \epsilon)(A_i + \beta) \).

Comparing with the original redundancy condition, slightly more instances will be kept.

Uncertainty score \( \beta_r \) is related to the inaccuracy of the instance \( A_r \). Notice that if an instance indexed by \( r \) is removed in interval \((i, j)\), we can approximate the interval’s uncertainty by \( A_i - A_j \leq \epsilon A_i \). Hence, we set \( \beta_r = \epsilon (A_i + \beta) \).

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.
foreach Feed the segment’s I-set to $S_i$.

10 if $l_1 = 0$ then Kill $A_{l_1}$, and $l \leftarrow l_1$
11 for $l \in 1 \text{ do } A_{l-1} \leftarrow A_l, l \leftarrow l - 1$
12 ReduceRedundancy() 

Procedure Process $(l, r)$

13 if $r \notin 1$ then
14 if $r$ has no successor then create a new $A_r$;
15 else
16 Let $i$ and $j$ denote $r$’s predecessor (or $i = 0$ if not exists) and successor, respectively;
17 $A_r \leftarrow$ a copy of $A_j$;
18 Find $a, b \in 1$ s.t. $(a, b) \geq (i, j)$, $\beta_r \leftarrow \beta_{ab}$;
19 Feed the segment’s I-set to $A_r, i \in [l, r]$;
20 ReduceRedundancy() 

foreach $i \in 1$ do

22 Find the largest $j > i$ s.t. $A_j \geq (1 - \epsilon)(A_i + \beta_i)$;
23 For each $l \in (i, j)$, kill $A_l$ and remove $l$ from $I$;
24 $\beta_{ij} \leftarrow \epsilon(A_i + \beta_i)$;

Algorithm 4 HistIT-Red

<table>
<thead>
<tr>
<th>dataset</th>
<th>nodes</th>
<th>activities</th>
<th>date range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>2.4M</td>
<td>64.6M</td>
<td>Jan/1 ~ 31/2016</td>
</tr>
<tr>
<td>Twitter</td>
<td>305K</td>
<td>563K</td>
<td>Jul/1 ~ 7/2012</td>
</tr>
<tr>
<td>Weibo</td>
<td>127K</td>
<td>1.5M</td>
<td>Jan/2012 ~ Dec/2012</td>
</tr>
</tbody>
</table>

• Reddit [41]. Reddit is a popular online forum where users can post subjects and comment to other users’ posts. If a user $u$ replies to another user $v$’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 $I_t$ 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 $\beta$ 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 $\epsilon$, 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 $\epsilon$, 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 $\lambda$. Parameter $\epsilon$ 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 $\hat{q}$ for approximating the true value $q$. NRMSE is defined by $NRMSE(\hat{q}) = \sqrt{\mathbb{E}[(\hat{q} - q)^2]} / q$.

• 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 $F_k(S)$ 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 $F_k(S)$ is a good estimate of $\mathbb{E}[f(\sigma(S; S_0, h))]$. To this end, we randomly pick $k = 10$ nodes as node set $S$, and then compute $F_k(S)$ using different sample size, as well as the true value $\mathbb{E}[f(\sigma(S; S_0, h))]$, on different datasets. Here, we use small $k$ for the convenience of computing the exact value of $\mathbb{E}[f(\sigma(S; S_0, h))]$. We then use the NRMSE to evaluate the accuracy of estimate $F_k(S)$. We vary $n$ from 10 to 500, and for each $n$, we vary $\lambda$ 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.
However, they indeed achieve similar solution quality as the solution quality of \textsc{HIST}, averaged after running 1 oracle calls ratio of \textsc{HIST}. We set \( \epsilon \) in range \([0.01, 0.02, 0.03, 0.04, 0.05]\), in order to expose the weakness of \textsc{BasicIT} when processing slowly decaying streams. We set \( \epsilon = 0.2, k = 10, \) and \( L = 100 \), i.e., the maximum lifespan in \textsc{BasicIT} is 100. We calculate two ratios: (1) the solution quality ratio, i.e., the quality of solution obtained by \textsc{HIST-RED} against the quality of solution obtained by \textsc{BasicIT}, and (2) the number of oracle calls ratio, i.e., the number of oracles of \textsc{HIST-RED} against the number of oracles of \textsc{BasicIT}. Both results are averaged after running 1,000 time steps, as shown in Fig. 9.

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

\begin{itemize}
  \item **Solution Quality.** To understand the solution quality of \textsc{HIST-RED}, next we compare the solution quality of \textsc{HIST-RED} with \textsc{Greedy} and \textsc{Random}, which serve as the upper and lower bound of solution quality, respectively.

  We run \textsc{HIST-RED} with fixed \( \lambda = 0.001 \) and varying \( \epsilon \) 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 \textsc{Greedy} algorithm always achieves the highest solution quality while the \textsc{Random} algorithm always achieves the lowest. In general, the solution quality achieved by \textsc{HIST-RED} is very close to \textsc{Greedy}, and is much better than \textsc{Random}. From the results, \( \epsilon \)’s effect seems to be unclear, and the curves overlap with each other. To further investigate how \( \epsilon \) affects the solution quality, we show the averaged solution quality ratio with respect to \textsc{Greedy} using different \( \epsilon \) on each dataset in Fig. 13(a). It is clear to see that when \( \epsilon \) is larger, the solution quality drops. As larger \( \epsilon \) means that fewer \textsc{SievePAIT} instances are maintained, thereby resulting in poorer solution quality.

  \item **Computational Efficiency.** Next we compare the computational efficiency of \textsc{HIST-RED} with \textsc{Greedy} in terms of the required number of oracle calls. Similar to the previous experiment, we run \textsc{HIST-RED} with fixed \( \lambda = 0.001 \) and varying \( \epsilon \) 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 \textsc{HIST-RED} requires much less oracle calls than \textsc{Greedy}, and \textsc{Greedy} is only efficient at the very
beginning of each stream (when the number of nodes is small) but the number of oracle calls increases quickly over time. For HISTIT-R, when \( \epsilon \) increases, the number of oracle calls further drops. Hence, HISTIT-R is more efficient when \( \epsilon \) 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 \( \epsilon \) has a trade-off effect in HISTIT-R, i.e., smaller \( \epsilon \) 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-R with other baselines. Here we fix \( \lambda = 0.001 \) and \( \epsilon = 0.2 \) for HISTIT-R. 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 processing streaming data, their throughput is significantly lower than HISTIT-R. This demonstrates the superiority of HISTIT-R 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], [18], 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 $X_i \triangleq \frac{1}{n} f(\sigma(S; S_i^{(i)}))$ for $i = 1, \ldots, n$. Then, $X_i \in [0, \frac{1}{n} f(\sigma(S; S_i))]$ and $F_i(S) = \sum_{i=1}^{n} X_i$. By Hoeffding’s inequality, we have

$$P \left( |F_i(S) - \mathbb{E}[f(\sigma(S; S_i; h))]| \geq \epsilon \mathbb{E}[f(\sigma(S; S_i; h))] \right) \leq 2 \exp \left( -\frac{2\epsilon^2 \mathbb{E}[f(\sigma(S; S_i; h))]^2}{f(\sigma(S; S_i^{(i)}))^2} \right) = 2 \exp \left( -\frac{2\epsilon^2 \rho^2}{f(\sigma(S; S_i^{(i)}))^2} \right)$$

where $\rho \triangleq \mathbb{E}[f(\sigma(S; S_i; h))]/f(\sigma(S; S_i))$. If we expect the above probability is no larger than $\delta$, 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 $F_i(S)$. For any sets $S \subseteq T \subseteq V$, by the definition of influence spread, we have $\sigma(S; S_i^{(i)}) \subseteq \sigma(T; S_i^{(i)})$. It follows that

$$F_i(S) = \frac{1}{n} \sum_{i=1}^{n} f(\sigma(S; S_i^{(i)})) \leq \frac{1}{n} \sum_{i=1}^{n} f(\sigma(T; S_i^{(i)})) = F_i(T)$$

where the inequality holds due to the monotonicity of $f$. Next we prove the submodularity of $F_i(S)$, i.e., for any sets $S \subseteq T \subseteq V$ and user $u \in V \setminus T$, we want to show

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

It follows that

$$F_i(S \cup \{u\}) - F_i(S) = \frac{1}{n} \sum_{i=1}^{n} \left[ f(\sigma(S \cup \{u\}; S_i^{(i)}) - f(\sigma(S; S_i^{(i)})) \right]$$

$$\geq \frac{1}{n} \sum_{i=1}^{n} \left[ f(\sigma(S; S_i^{(i)}) \cup \sigma(u; S_i^{(i)})) - f(\sigma(S; S_i^{(i)})) \right].$$

Because $f$ is a submodular function, hence

$$f(\sigma(S; S_i^{(i)}) \cup \sigma(u; S_i^{(i)})) - f(\sigma(S; S_i^{(i)})) \geq f(\sigma(T; S_i^{(i)}) \cup \sigma(u; S_i^{(i)})) - f(\sigma(T; S_i^{(i)})).$$

It follows that

$$F_i(S \cup \{u\}) - F_i(S) \geq \frac{1}{n} \sum_{i=1}^{n} \left[ f(\sigma(T; S_i^{(i)}) \cup \sigma(u; S_i^{(i)})) - f(\sigma(T; S_i^{(i)})) \right]$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[ f(\sigma(T \cup \{u\}; S_i^{(i)})) - f(\sigma(T; S_i^{(i)})) \right]$$

$$= F_i(T \cup \{u\}) - F_i(T).$$

We thus conclude that $F_i$ is a monotone submodular function. □

PROOF OF LEMMA 4

Proof: In incremental sampling, a social activity $a \in S_{i-1}^{(i)}$ is included in $S_{i}^{(i)}$ 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 $S_{i}^{(i)}$, and state 0 denotes not. The transition matrix of this Markov chain is $P = [1-e^{-\lambda} e^{\lambda}]$. Consider a social activity $a$ arrived at time $t_0$, its state can be represented by $\pi_{t_0} = [1-p_0, p_0]$, i.e., with probability $p_0$, $a$ is included in a sample. Then at time $t$, we can obtain that $\pi_{t} = \pi_{t_0} P^{t-t_0} = [1-p_0 e^{-\lambda(t-t_0)}, p_0 e^{-\lambda(t-t_0)}]$. It means that, at time $t$, the element is included in $S_{i}^{(i)}$ with probability $p_0 e^{-\lambda(t-t_0)} = p_0(\lambda)$. □

PROOF OF THEOREM 1

Proof: Let $S_{i}^{*} = \arg \max_{S \subseteq V \setminus \{S_i^{(i)}\} \subseteq k} F_i(S)$, OPT$_i \triangleq F_i(S_{i}^{*})$, and $\theta^* = OPT_i/2k$. We further define $\Delta_i \triangleq \max_{a \in V} F_i(\{a\})$. It is easy to observe that $\max\{\Delta_i, LB\} \leq OPT_i \leq k \Delta_i$ and there is a threshold $\theta$ such that $(1-\epsilon)\theta^* \leq \theta < \theta^*$. Let $S_{\theta,t}$ denote the set of users corresponding to threshold $\theta$ maintained in SIEVEPAIT at time $t$. We partition set $S_{\theta,t}$
into two disjoint subsets $S_{θ,t-1}$ and $S'_{θ,t}$ where $S_{θ,t-1}$ is the 
 maintained users at previous time step $t-1$ and $S'_{θ,t} \subseteq V_a$ is 
 the set of users newly selected from $V_a$ at time $t$. Our goal is 
 to show that $F_t(S_{θ,t}) \geq (1/2 - ε)OPT_t$.

We first inductively show that $F_t(S_{θ,t}) \geq |S_{θ,t}|θ$. Thus if 
 $|S_{θ,t}| = k$, then $F_t(S_{θ,t}) \geq kθ \geq (1 - ε)OPT_t/2$.

At time $t = 1$, by definition, $S_{θ,1} = S_{θ,1}$ consists of users such 
 that their utility gains are at least $θ$. Therefore $F_1(S_{θ,1}) \geq 
 |S_{θ,1}|θ$. Assume $F_{t-1}(S_{θ,t-1}) \geq |S_{θ,t-1}|θ$ at time $t - 1$. Let 
 us consider $F_t(S_{θ,t})$ at time $t$.

$$F_t(S_{θ,t}) = F_t(S_{θ,t-1} \cup S'_{θ,t}) = F_t(S_{θ,t-1} \setminus S'_{θ,t}) - F_t(S_{θ,t-1}) + F_t(S'_{θ,t-1})$$

The first part corresponds to the gain of newly selected users $S'_{θ,t}$ 
 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 $|S'_{θ,t}|θ$.

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

We thus obtain $F_t(S_{θ,t}) \geq |S'_{θ,t}| + |S_{θ,t-1}|θ = |S_{θ,t}|θ$. 
 Therefore, $F_t(S_{θ,t}) \geq (1 - ε)/2 \cdot OPT_t$ when $|S_{θ,t}| = k$.

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

$$OPT_t - F_t(S_{θ,t}) \leq \sum_{u \in S_{θ,t} \setminus S_{θ,t}} Δ(u|S_{θ,t})$$

$$= \sum_{u \in S_{θ,t} \setminus S_{θ,t}} Δ(u|S_{θ,t}) + \sum_{v \in S_{θ,t} \setminus S_{θ,t}} Δ(v|S_{θ,t})$$

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

$$Δ(v|S_{θ,t-1}) = F_t(\{v\} \cup S_{θ,t-1}) - F_t(S_{θ,t-1}) \leq F_{t-1}(\{v\} \cup S_{θ,t-1}) - F_{t-1}(S_{θ,t-1}) \leq θ.$$

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

$$OPT_t - F_t(S_{θ,t}) \leq kθ \leq k(1 + ε)OPT_t = \frac{1 + ε}{2}OPT_t,$$

which implies $F_t(S_{θ,t}) \geq (1 - ε)OPT_t/2$.

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

**Proof of Theorem 2**

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

- **Update complexity.** For each user, we need to calculate 
  the utility gain $|Θ|$ times (with respect to each $S_{θ}$) resulting 
  in $O(\epsilon^{-1} log k)$ 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 $O(γnε^{-1} log k)$.

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

**Proof of Lemma 5**

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

Otherwise, $A_{l-1}$ and $A_{l+1}$ 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 $A'_{l+1} \geq (1 - ε)A'_{l}$ 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 $A_l$ and $A_{l+1}$ 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 $A_0$ exists. 
 Because $A_0$’s input is exactly $T_l$, hence we have

$$A_1 = A_0 \geq 1 - εOPT_t.$$

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

Note that indexes in $I_l$ 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 $A'_{l-1}$ and $A'_{l}$ since they were 
  created. In other words, instance $A_{l}$ has the same inputs 
  as $A_{0}$ because their input I-sets always belong to the same 
  segment. Hence, $A_{l_1}$ and $A_{0}$ have the same output at time $t$.

So we have

$$A_{l_1} = A_0 \geq 1 - εOPT_t.$$
and from \( t' \) to \( t \), no instance is created between \( A'_{t_0} \) and \( A'_{t_1} \).

Let us clarify the relation of the inputs of three SIEVEPAIT instances \( A_{t_0}, A_0, \) and \( A_{t_1} \). Note that at time \( t \), only \( A_{t_1} \) is practically maintained, while the other two actually do not exist. At time \( t' \), let \( I_1, I_2, \) and \( I_3 \) denote the processed I-sets of \( A'_{t_0}, A'_{t_1}, \) and \( A'_{t_1} \), respectively, as illustrated in Fig. 16. From \( t' \) to \( t \), as claimed in Case 2 of Lemma 5, the I-sets fed to \( A_0 \) and \( A_{t_1} \) always belong to the same segment. Hence, \( A_0 \) and \( A_{t_1} \) have the same inputs in time interval \((t', t] \), denoted by \( I_0 \), as illustrated in Fig. 16. At time \( t \), instance \( A_0 \) has processed I-sets \( I_2 || I_0 \), and \( A_{t_1} \) has processed I-sets \( I_3 || I_0 \), where \( |\cdot| \) denotes the concatenation of two I-set streams. We want to find out how close the output of \( A_{t_1} \) is to that of \( A_0 \).

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

At time \( t \), we have

\[
A_{t_1} = A(I_3 || I_0) = \frac{1}{2} (A(I_3 || I_0) + A(I_3 || I_0))
\geq \frac{1}{2} (A(I_3) + A(I_3 || I_0))
\geq \frac{1}{2} ((1 - \epsilon)A(I_1) + A(I_3 || I_0))
\geq \frac{1}{2} - \epsilon (A(I_1) + A(I_3 || I_0))
\]

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 \( A(I_1) \geq (1 - \epsilon)/2 \cdot OPT(I_1) \) and \( A(I_3 || I_0) \geq (1 - \epsilon)/2 \cdot OPT(I_3 || I_0) \). It follows that

\[
A_{t_1} \geq \frac{1}{4} (OPT(I_1) + OPT(I_3 || I_0)).
\]

Let \( S_{t_1}^* \) denote an optimal user set in I-sets \( I_2 || I_0 \), i.e., \( F(S_{t_1}^*; I_2 || I_0) = OPT(I_2 || I_0) = OPT_1 \). Since \( OPT(I_1) \) is the optimal utility in \( I_1 \), we have

\[
OPT(I_1) \geq F(S_{t_1}^*; I_1).
\]

Note that \( I_3 \) contains more social activities than \( I_2 \), hence

\[
F(S_{t_1}^*; I_3) \geq F(S_{t_1}^*; I_2).
\]

Since \( OPT(I_3 || I_0) \) is the optimal utility in \( I_3 || I_0 \), hence

\[
OPT(I_3 || I_0) \geq F(S_{t_1}^*; I_3 || I_0).
\]

It follows that

\[
OPT(I_1) + OPT(I_3 || I_0)
\geq F(S_{t_1}^*; I_2 || I_0) + F(S_{t_1}^*; I_3 || I_0)
\geq \frac{1}{n} \sum_{i=1}^{n} [f(\sigma(S_{t_1}^*; I_2(i))) + f(\sigma(S_{t_1}^*; I_3 || I_0))]
\geq \frac{1}{n} \sum_{i=1}^{n} [f(\sigma(S_{t_1}^*; I_2(i))) \cup \sigma(S_{t_1}^*; I_3 || I_0))]
\geq \frac{1}{n} \sum_{i=1}^{n} [f(\sigma(S_{t_1}^*; I_2 || I_0))]
= F(S_{t_1}^*; I_2 || I_0) = OPT(I_2 || I_0) = OPT_t,
\]

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

\[
A(I_3 || I_0) \geq \frac{1}{4} (1 - \epsilon)OPT_1 \geq (1 - \epsilon)OPT_t,
\]

which completes the proof.

\[\square\]

**Proof of Theorem 6**

**Proof:** The reducing redundancy operation ensures that \( A_{t_{i+2}} < (1 - \epsilon)A_{t_i} \) always holds. Because \( A_t \in [\Delta, k\Delta] \) where \( \Delta \triangleq \max_i F_t(u) \), then \( |I_t| \) is upper bounded by \( O(\log(1 - \epsilon) - k) = O(\epsilon^{-1} \log k) \).

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

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

\[\square\]

**References**


