Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

8-2017

Real-time influence maximization on dynamic social streams

Yanhao WANG
National University of Singapore

Qi FAN
National University of Singapore

Yuchen LI
Singapore Management University, yuchenli@smu.edu.sg

Kian-Lee TAN
National University of Singapore

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Databases and Information Systems Commons, and the Social Media Commons

Citation

WANG, Yanhao; FAN, Qi; LI, Yuchen; and TAN, Kian-Lee. Real-time influence maximization on dynamic social streams. (2017). *Proceedings of the VLDB Endowment: 43rd International Conference on Very Large Data Bases, Munich, Germany, 2017 August 28 - September 1*. 10, 805-816.
Available at: https://ink.library.smu.edu.sg/sis_research/4027

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Real-Time Influence Maximization on Dynamic Social Streams

Yanhao Wang, Qi Fan, Yuchen Li, Kian-Lee Tan
School of Computing, National University of Singapore, Singapore
{yanhao90, fanqi, liyuchen, tankl}@comp.nus.edu.sg

ABSTRACT

Influence maximization (IM), which selects a set of k users (called seeds) to maximize the influence spread over a social network, is a fundamental problem in a wide range of applications such as viral marketing and network monitoring. Existing IM solutions fail to consider the highly dynamic nature of social influence, which results in either poor seed qualities or long processing time when the network evolves. To address this problem, we define a novel IM query named Stream Influence Maximization (SIM) on social streams. Technically, SIM adopts the sliding window model and maintains a set of k seeds with the largest influence value over the most recent social actions. Next, we propose the Influential Checkpoints (IC) framework to facilitate continuous SIM query processing. The IC framework creates a checkpoint for each window shift and ensures an ε-approximate solution. To improve its efficiency, we further devise a Sparse Influential Checkpoints (SIC) framework which selectively keeps $O(\frac{\log N}{\beta})$ checkpoints for a sliding window of size N and maintains an $\frac{\varepsilon(1-\beta)}{2}$-approximate solution. Experimental results on both real-world and synthetic datasets confirm the effectiveness and efficiency of our proposed frameworks against the state-of-the-art IM approaches.

1. INTRODUCTION

Social media advertising has become an indispensable tool for many companies to promote their business online. Such trends have generated 26.89 billion dollars advertising revenue for Facebook in 2016[1]. *Influence Maximization* (IM) is a key algorithmic problem behind social media viral marketing [13,18]. Through the word-of-mouth propagation among friends, IM aims to select a set of k users such that the source information (e.g., advertisement) is maximally spread in the network, and it has been extensively researched [2,7,10,11,16–18,22,24–26,29,30,35–38] in the last decade. Besides viral marketing, IM is also the cornerstone in many other important applications such as network monitoring [21] and recommendation [33].

Most existing IM solutions rely on influence probabilities between users to acquire the seeds. The influence probabilities are usually derived from social actions in online social networks [15,20], e.g., "retweets" on Twitter. In reality, social influences are highly dynamic and the propagation tendencies between users can be altered drastically by breaking news and trending topics. Consequently, the seeds selected by IM methods [7,18,29,35,36] that assume static social influences can quickly become outdated. Recently, there are some research efforts on IM under dynamic social influences. However, existing solutions for dynamic IM either cannot offer theoretical guarantees for the qualities of selected seeds (e.g., [2,38]) or provide guarantees at the expense of high processing overhead for updates (e.g., [11,30]). In fact, the state-of-the-art dynamic IM solution [30] can only process several hundred updates per second, which is far lower than the update rates of real-world social networks, e.g., about 7,500 tweets are generated on Twitter per second[2].

To resolve the aforementioned drawbacks and make dynamic IM both effective and efficient, we propose a novel *Stream Influence Maximization* (SIM) query to track influential users in real-time. SIM utilizes the widely available social actions to estimate the social influences and maintain the seed set continuously. To capture the short-term memory effect of social influences [32], where past influences quickly fade as new influences emerge, SIM adopts the *sliding window* model [12] which always considers the most recent N actions and strives to find k users who collectively have the largest influence value in the current window. In addition, SIM supports general monotone submodular functions to compute the influence values as such functions are often used to represent the "diminishing returns" property of social influences in different types of IM problems [5,10,16,18,22,24,37].

Due to the NP-hardness of SIM, we focus on processing it approximately with theoretical bounds. Leveraging the monotonicity and submodularity of influence functions, a naïve greedy algorithm [28] can provide a $(1-1/e)$ approximate solution for SIM. However, the greedy algorithm requires $O(k\cdot|U|)$ ($|U|$ is the number of users in the network) influence function evaluations for each update. Empirically, it takes around 10 seconds to select 100 seeds from a network with $500,000$ users, which hardly matches the rates of real-world social streams. Another closely related technique to SIM is *Streaming Submodular Optimization* (SSO) [4,19].

[1] https://investor.fb.com/financials/default.aspx

[2] http://www.internetlivestats.com/one-second/

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.

Proceedings of the VLDB Endowment, Vol. 10, No. 7
Copyright 2017 VLDB Endowment 2150-8097/17/03.

Existing SSO approaches [4, 19] can provide solutions with theoretical guarantees for maximizing submodular functions with cardinality constraints over append-only streams. However, to the best of our knowledge, none of the proposed SSO algorithms can support the sliding window model.

In this paper, we propose a novel *Influential Checkpoints* (IC) framework to support efficient SIM query processing with theoretical bounds. IC not only tracks the solution for the current window but also maintains partial solutions called *Checkpoints* for future windows that overlap with the current window. Therefore, for every subsequent window shift, the up-to-date solution can be retrieved efficiently. We further design a generic Set-Stream Mapping (SSM) interface which can adapt many existing SSO algorithms to SIM so that the solution retrieved for each window has at least the same approximation ratio as those algorithms. However, maintaining all $O(N)$ checkpoints incurs significant update overhead (N is the number of actions in a window). To support efficient IC maintenance, we propose a *Sparse Influential Checkpoints* (SIC) framework to selectively maintain a subset of checkpoints by leveraging the monotonicity and subadditivity of the influence values returned by different checkpoints. Consequently, SIC only keeps $O(\frac{\log N}{\beta})$ checkpoints and maintains an $\frac{\varepsilon(1-\beta)}{2}$-approximate solution.

We hereby summarize our contributions as follows.

- We address the limitations of existing IM solutions in supporting fast evolving social networks and propose a novel SIM query over sliding windows. (Section 3)
- We develop a novel *Influential Checkpoints* (IC) framework for SIM query processing. It is integrated with a generic Set-Stream Mapping (SSM) interface to incorporate existing ε-approximate SSO algorithms while retaining their approximation ratios. (Section 4)
- We further propose the SIC framework to selectively maintain $O(\frac{\log N}{\beta})$ checkpoints for a sliding window of size N. Leveraging the subadditivity and submodularity of the influence values returned by different checkpoints, an $\frac{\varepsilon(1-\beta)}{2}$-approximation ratio is always guaranteed. (Section 5)
- We experimentally evaluate the effectiveness and efficiency of our proposed frameworks. First, the qualities of the seeds selected by IC and SIC are competitive with the state-of-the-art IM algorithms in both static and dynamic settings. Second, SIC achieves speedups of up to 2 orders of magnitude over the static approaches. Third, SIC achieves up to 8 times speedups over IC with less than 5% quality losses. (Section 6)

2. RELATED WORK

We summarize the most relevant literature from three areas: influence maximization, streaming submodular optimization and function estimation on sliding windows.

2.1 Influence Maximization (IM)

IM aims to extract a given number of users that maximize the influence spread over a network. Previous efforts on IM can be generally categorized into static methods and dynamic methods based on their abilities to handle changes in social influences. Here, we summarize them separately.

IM in Static Networks: There has been a vast amount of literature on influence maximization (IM) in static networks over the last decade (see [7, 10, 18, 22, 24, 26, 29, 35–37]). The state-of-the-art static IM method on the classic influence models (i.e., independent cascade (IC) and linear threshold (LT)) is IMM [35]. It runs in nearly linear time wrt. the graph size with a $(1 - 1/e - \varepsilon)$ approximation guarantee. Nevertheless, static IM methods including IMM cannot efficiently support highly evolving networks since a complete rerun is required for every update on influence graphs.

There are also many static methods considering different types of IM problems by extending classic influence models. For example, topic-aware IM [5, 10] considers the influence diffusion under topic models; location-aware IM [22, 37] focuses on maximizing the influence spread in certain spatial areas; and conformity-aware IM [24] considers users' conformity tendencies in the influence estimation.

IM in Dynamic Networks: Recently, there are emerging studies about IM in dynamic networks. However, most of these methods cannot provide a theoretical guarantee of their seed quality and may return arbitrarily bad solutions [2, 38]. Chen et al. [11] proposed an Upper Bound Interchange (UBI) method with a 1/2-approximation ratio. Nevertheless, UBI is sensitive to the number of users selected. When the size of the seed set increases, both its performance and solution quality degrade dramatically. This prevents UBI from being practically useful. Very recently, a new dynamic IM method with a theoretical bound is presented in [30]. It dynamically maintains a RIS-based [7] index against changes on graphs and achieves a $(1 - 1/e - \varepsilon)$ approximation ratio. However, due to the high maintenance cost, it can only process several hundred of influence graph updates per second, which cannot meet the requirement of real-world social streams. Therefore, existing dynamic IM methods cannot provide high-quality solutions efficiently.

2.2 Streaming Submodular Optimization

Another closely related field to SIM is the Streaming Submodular Optimization (SSO) [3, 4, 19, 31]. SSO adopts the append-only streaming model where elements arrive one by one and the objective is to dynamically maintain a set of at most k elements to maximize a submodular function wrt. all the observed elements at any time. Saha et al. [31] and Ausiello et al. [3] developed two approaches for a special case of SSO (i.e., the online Maximum k-Coverage problem) with the same 1/4 approximation ratio. The state-of-the-art SSO solutions are SieveStreaming [4] and ThresholdStream [19], both of which achieve a $(1/2 - \beta)$ approximation ratio. Unfortunately, SSO algorithms cannot be directly applied to the sliding window model because they do not handle the continuous expiry of elements. Nevertheless, we will show in Section 4.2 that existing SSO algorithms can serve as checkpoint oracles in the *IC* and *SIC* frameworks.

2.3 Function Estimation on Sliding Windows

Several works [8, 12] studied how to continuously estimate a function in the sliding window model. They leverage special properties of target functions to achieve sublinear performance and reasonable quality. Let g be the target function, and A,B,C be three sequences on streams such that B is a tail subsequence of A and C is contiguous to B. The exponential histogram [12] is proposed to approximate *weakly additive* functions, i.e., $g(A) + g(C) \leq g(A \cup C) \leq c(g(A) + g(C))$ for some small constant c. The smooth histogram [8] requires that the target functions are (α, β)-*smooth*. Specifically, we say g is (α, β)-*smooth* if $\frac{g(B)}{g(A)} \geq$

$1-\beta$, then $\frac{g(B\cup C)}{g(A\cup C)} \geq 1-\alpha$ for some $0 < \beta \leq \alpha < 1$. Following the analysis in [8], smooth histograms are applicable only when g can be computed with an approximation ratio of at least 0.8 over append-only streams. In this paper, we adopt monotone submodular influence functions [18] widely used in the social influence analysis. However, such functions are not *weakly additive* and existing SSO algorithms [4, 19] cannot provide a more than 1/2 approximate solution over append-only streams. This implies that both techniques cannot be directly applied to our scenario.

3. PROBLEM STATEMENT

We consider a social stream over a social network with a user set U. The social stream comprises unbounded time-sequenced social actions which are generated by user activities. Let $a_t = \langle u, a_{t'}\rangle_t$ $(t' < t)$ be an action at time t representing the following social activity: *user u performs a_t at time t responding to an earlier action $a_{t'}$*. Typical actions include "retweet" on Twitter, "reply" on Reddit, "comment" on Facebook, to name just a few. If an action a_t does not respond to any previous action, e.g., a user u posted an original tweet, we call it a *root action* and denote it by $a_t = \langle u, nil\rangle_t$.

Like many data streams, social streams are time-sensitive: recent actions are more valuable than those in the past. We adopt the well-recognized *sequence-based sliding window* [12] model to capture such essence. Let N be the window size, a sequence-based sliding window W_t maintains the latest N actions till a_t in the stream, i.e., $W_t = \{a_{t-N+1}, \ldots, a_t\}$. For simplicity, we use $W_t[i]$ to represent the i-th $(i \geq 1)$ action within W_t. Then, we use $A_t \subseteq U$ to denote the set of active users who perform at least one action in W_t, i.e., $A_t = \{W_t[i].u | i = 1, \ldots, N\}$.

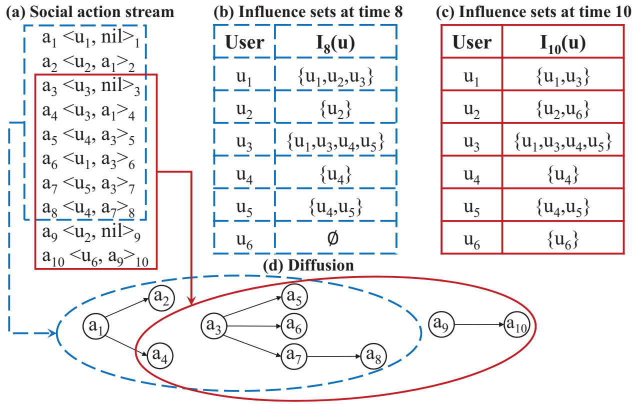
Since social actions directly reflect the information diffusion in the social network [15, 16, 20, 34], we define the *influence* between users according to their performed actions. We say user u *influences* user v in W_t, denoted by $(u \rightsquigarrow v)_t$, if there exists an action a performed by user v s.t. $a \in W_t$ and a is *directly* or *indirectly* triggered by an action a' of u. It is notable that such an a' is not necessarily in W_t.

We formally define the influence set of a user as follows:

Definition 1. *The influence set of a user $u \in U$ at time t, denoted as $I_t(u) \subseteq A_t$, is the set of users who are influenced by u wrt. the sliding window at time t (i.e., W_t). Equivalently, $I_t(u) = \{v | (u \rightsquigarrow v)_t\}$.*

Intuitively, the influence set of u denotes the set of users who recently performed actions under the impact of u. The concept of the influence set can be naturally extended to a set of users. In particular, let $S = \{u_1, \ldots, u_k\}$ be a set of k users, the influence set of S wrt. W_t is a union of the influence sets of all its members, i.e., $I_t(S) = \cup_{u\in S} I_t(u)$. Then the influence value of S is measured by $f(I_t(S)) : 2^{|U|} \rightarrow \mathbb{R}_{\geq 0}$. We consider $f(I_t(\cdot))$ as a nonnegative monotone[3] submodular[4] function [28] which is widely adopted by many IM problems for its natural representation of the "diminishing returns" property on the social influence [18].

For ease of presentation, we only consider the cardinality function, i.e., $f(I_t(\cdot)) = |I_t(\cdot)|$, as the influence function in the remaining of this paper. It should be noted that any other monotone submodular influence functions can also be used in our frameworks.


(a) Social action stream
$a_1 \langle u_1, nil\rangle_1$
$a_2 \langle u_2, a_1\rangle_2$
$a_3 \langle u_3, nil\rangle_3$
$a_4 \langle u_3, a_1\rangle_4$
$a_5 \langle u_4, a_3\rangle_5$
$a_6 \langle u_1, a_3\rangle_6$
$a_7 \langle u_5, a_3\rangle_7$
$a_8 \langle u_4, a_7\rangle_8$
$a_9 \langle u_2, nil\rangle_9$
$a_{10} \langle u_6, a_9\rangle_{10}$
(b) Influence sets at time 8
(c) Influence sets at time 10
(d) Diffusion
a_1
a_2
a_4
a_3
a_5
a_6
a_7
a_8
a_9
a_{10}

User	$I_8(u)$
u_1	$\{u_1,u_2,u_3\}$
u_2	$\{u_2\}$
u_3	$\{u_1,u_3,u_4,u_5\}$
u_4	$\{u_4\}$
u_5	$\{u_4,u_5\}$
u_6	$\emptyset$

User	$I_{10}(u)$
u_1	$\{u_1,u_3\}$
u_2	$\{u_2,u_6\}$
u_3	$\{u_1,u_3,u_4,u_5\}$
u_4	$\{u_4\}$
u_5	$\{u_4,u_5\}$
u_6	$\{u_6\}$

Figure 1: A social action stream and the influences of users over the sliding windows.

Example 1 illustrates our definition of *influence* over the sliding windows on a social action stream.

Example 1. Figure 1(a) gives an example of a social action stream. Given the window size $N = 8$, two windows, W_8 and W_{10}, are highlighted in blue and red boxes respectively. In Figure 1(b), $I_8(u_1) = \{u_1, u_2, u_3\}$ as a_1,a_6 are performed by u_1 and a_2,a_4 performed by u_2,u_3 respectively are triggered by a_1 in W_8. When the window shifts from W_8 to W_{10}, a_1,a_2 expire while a_9,a_{10} arrive. Then, $I_{10}(u_1) = \{u_1, u_3\}$ as Figure 1(c). Due to the expiry of a_2, u_2 is deleted from $I_{10}(u_1)$. However, since a_4 has not expired yet, u_1 still *influences* u_3 in W_{10} regardless of the expiry of a_1.

As new actions arrive at high speed while old ones expire at the same rate, users with the largest influence values keep evolving. To track the influential users over social streams in real-time, we propose a **S**tream **I**nfluence **M**aximization (SIM) query which is formally defined as follows:

Definition 2. *Let W_t be the sliding window at time t, Stream Influence Maximization (SIM) is a* continuous *query on a social stream that returns a set of at most k users S_t^{opt} who collectively achieve the largest influence value wrt. W_t:*

$$S_t^{opt} = \operatorname{argmax}_{S\in U \wedge |S| \leq k} f(I_t(S)) \tag{1}$$

We continue with the running example in Figure 1 to show how SIM keeps track of the most influential users over the sliding windows.

Example 2. Given $k = 2$, SIM returns $S_8^{opt} = \{u_1, u_3\}$ as the most influential users at time 8 since $I_8(S_8^{opt}) = I_8(u_1) \cup I_8(u_3)$ contains all users in A_8. We have $f(I_8(S_8^{opt})) = 5$ using the cardinality function. However, as a_1,a_2 expire while a_9,a_{10} arrive at time 10, $f(I_{10}(S_8^{opt})) = 4$ as u_2 is deleted from $I_{10}(S_8^{opt})$. Thus, SIM returns $S_{10}^{opt} = \{u_2, u_3\}$ in W_{10} because $I_{10}(S_{10}^{opt})$ contains all users in A_{10}. We have $f(I_{10}(S_{10}^{opt})) = 6$ accordingly.

Note that the solutions proposed in this paper also support the case where the sliding window shifts for more than one action. For simplicity, we focus on presenting solutions for handling sliding windows with a single action shift at a

[3] A set function g is monotone if for all $A \subseteq B$, $g(A) \leq g(B)$.
[4] A set function g is submodular if for all $A \subseteq B$, and any element $x \notin B$, $g(A \cup \{x\}) - g(A) \geq g(B \cup \{x\}) - g(B)$.

Table 1: Frequently used notations

Symbol	Definition and Description
U	the set of all users in a social network
$a_t = \langle u, a_{t'} \rangle_t$	a user u performs an action at time t triggered by an action $a_{t'} (t' < t)$
N	the size of the sliding window
L	the number of actions for each window shift, $L = 1$ by default
$W_t, W_t[i]$	the sliding window at time t, and the i-th action in the window
$I_t(u), I_t(S)$	the influence set of a user u or a set of users S wrt. W_t
$I_t[i](u), I_t[i](S)$	the influence set of u or S for contiguous actions $\{W_t[i], \dots, W_t[N]\}$
f	a monotone submodular influence function
k	the cardinality constraint of SIM
$\Lambda_t[i]$	an influential checkpoint maintaining an ε-approximate solution of SIM for $\{W_t[i], \dots, W_t[N]\}$
$S_t^{opt}, \mathsf{OPT}_t$	the optimal seed set of SIM wrt. W_t, and its influence value $\mathsf{OPT}_t = f(I_t(S_t^{opt}))$
$S_t^{opt}[i], \mathsf{OPT}_t[i]$	the optimal seed set of SIM for $\{W_t[i], \dots, W_t[N]\}$, and its influence value

time and leave the discussion on handling multiple action shifts at a time to Section 5.3.

It can be shown that SIM is NP-hard by reducing a well-known NP-hard problem, i.e., *Maximum k-Coverage* [3, 14, 31], to SIM in polynomial time. (see the proof of Theorem 1 in [1] for details).

Before moving on to the technical parts of this paper, we summarize the frequently used notations in Table 1.

4. INFLUENTIAL CHECKPOINTS

Since SIM is NP-hard, it is infeasible to maintain the optimal seed set for each sliding window in polynomial time. Therefore, our goal is to maintain an approximate solution achieving a bounded ratio to the optimal one efficiently. A naïve scheme is to run the *greedy* algorithm [28] for each window shift. The *greedy* algorithm starts with an empty user set $S_0 = \emptyset$, and at each iteration i $(1 \le i \le k)$, it incrementally adds a user u to the partial user set S_{i-1} maximizing $f(I_t(S_{i-1} \cup \{u\})) - f(I_t(S_{i-1}))$. Although it guarantees a $(1 - 1/e)$ approximate solution, which is the best possible approximation ratio for submodular maximization with cardinality constraints [28], $O(k \cdot |U|)$ influence function evaluations are needed for each update. Such an inefficient update scheme makes the *greedy* algorithm unable to handle a large window size with new actions arriving at high speed.

A key challenge for efficiently supporting SIM over sliding windows is to handle the expiry of old actions and the arrival of new actions simultaneously. Such a compound update pattern brings about fluctuations on users' influence sets which potentially degrade the quality of previously maintained seeds. In the remaining of this section, we present a novel *Influential Checkpoints* (IC) framework, which consists of a sequence of *checkpoint oracles* to efficiently handle the expiry and the arrival of actions simultaneously. We first give an overview of the IC framework in Section 4.1. Then, we describe how to construct a checkpoint oracle in Section 4.2. Finally, we take SieveStreaming [4] as an example to illustrate the Set-Stream Mapping interface in Section 4.3.

4.1 The Influential Checkpoints Framework

The high level idea of the IC framework is to avoid handling the expiry of old actions when the window shifts. Towards this goal, the framework maintains a partial result (i.e., an influential checkpoint) incrementally for each window shift. When an old action expires, the outdated result is simply deleted. In this way, the sliding window model is transformed to a simpler *append-only* model for each checkpoint, where many existing approaches [4, 19] can provide theoretically bounded approximate solutions.

Technically, let an influential checkpoint $\Lambda_t[i]$ $(1 \le i \le N)$ denote a *checkpoint oracle*[5] which provides an ε-approximate solution for SIM over contiguous actions $\{W_t[i], \dots, W_t[N]\}$. By maintaining N checkpoints (i.e., $\Lambda_t[1], \dots, \Lambda_t[N]$), a simple procedure to handle a window shift from W_{t-1} to W_t is presented in Algorithm 1. Whenever a new action a_t arrives, the oldest checkpoint in W_{t-1} (i.e., $\Lambda_{t-1}[1]$) expires and a new checkpoint $\Lambda_t[N]$ is added to W_t (Line 2). After adding the remaining checkpoints in W_{t-1} to W_t (Lines 3-4), each checkpoint in W_t processes a_t as an appending action to update its partial solution (Lines 5-6). To answer the SIM query for W_t, we simply return the solution of $\Lambda_t[1]$.

Algorithm 1 IC Maintenance

```
Require: IC:$\{\Lambda_{t-1}[1], \dots, \Lambda_{t-1}[N]\}$
1: — on receiving action $a_t$ —
2: Delete $\Lambda_{t-1}[1]$, create $\Lambda_t[N]$;
3: for all $\Lambda_{t-1}[i]$ do
4:     $\Lambda_t[i-1] \leftarrow \Lambda_{t-1}[i]$;
5: for all $\Lambda_t[i]$ do
6:     $\Lambda_t[i].process(a_t)$;
7: — on query —
8: return the solution of $\Lambda_t[1]$;
```

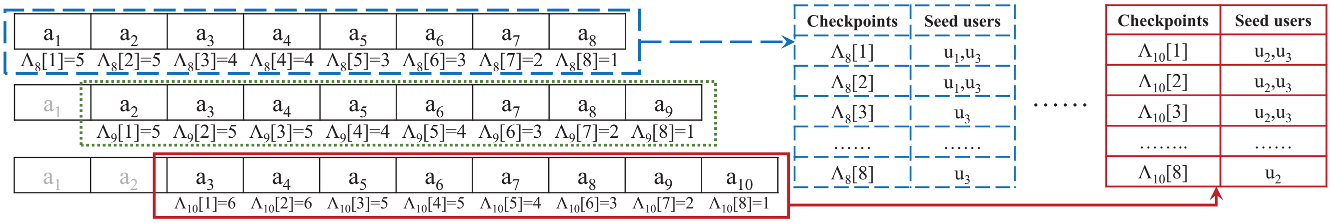
It is not hard to see that once each checkpoint oracle maintains an ε-approximate solution for its *append-only* action stream, IC always returns the solution with the same approximation ratio.

Example 3. Figure 2 illustrates the maintenance of checkpoints in the IC framework following Example 1. Let $N = 8$ and $k = 2$. The number of checkpoints always equals to the window size (i.e., 8). When action a_{10} arrives, a_2 will expire. Consequently, the checkpoint $\Lambda_{10}[8]$ is created and $\Lambda_9[1]$ is deleted. When the SIM query is issued at time 10, the solution of $\Lambda_{10}[1]$ (i.e.,$\{u_2, u_3\}$) is returned.

4.2 Checkpoint Oracle

The approximation ratio of IC relies on the checkpoint oracle providing an ε-approximate solution over an *append-only* action stream. Although submodular maximization in an *append-only* stream has attracted many research interests [3,4,19,31], they focused on a different set-stream model where elements in the stream are sets instead of actions. In general, an algorithm $\mathcal{A}$ on an append-only set-stream contains two components: $f'(\cdot)$ is a monotone submodular objective function and CX_t is a candidate solution containing no more than k sets from t observed sets (i.e., $X_1, \dots, X_t$). Given a stream of sets $\{X_1, X_2, \dots, X_m\}$, the objective of $\mathcal{A}$ is to maximize $f'(CX_t)$ at any time t $(1 \le t \le m)$. Although this problem resembles our problem, the set-stream

[5] We overload the notation $\Lambda_t[i]$ to denote the influence value returned by the oracle when it is clear in the context.


a1 a2 a3 a4 a5 a6 a7 a8
Λ8[1]=5 Λ8[2]=5 Λ8[3]=4 Λ8[4]=4 Λ8[5]=3 Λ8[6]=3 Λ8[7]=2 Λ8[8]=1
a1 a2 a3 a4 a5 a6 a7 a8 a9
Λ9[1]=5 Λ9[2]=5 Λ9[3]=5 Λ9[4]=4 Λ9[5]=4 Λ9[6]=3 Λ9[7]=2 Λ9[8]=1
a1 a2 a3 a4 a5 a6 a7 a8 a9 a10
Λ10[1]=6 Λ10[2]=6 Λ10[3]=5 Λ10[4]=5 Λ10[5]=4 Λ10[6]=3 Λ10[7]=2 Λ10[8]=1
……

Checkpoints	Seed users
$\Lambda_8[1]$	u_1,u_3
$\Lambda_8[2]$	u_1,u_3
$\Lambda_8[3]$	u_3
……	……
$\Lambda_8[8]$	u_3

Checkpoints	Seed users
$\Lambda_{10}[1]$	u_2,u_3
$\Lambda_{10}[2]$	u_2,u_3
$\Lambda_{10}[3]$	u_2,u_3
……..	……
$\Lambda_{10}[8]$	u_2

Figure 2: An example of checkpoint maintenance in IC.

model cannot directly fit in our scenario due to the following mismatch: it strives to keep k *sets* from a *stream of sets* but all observed sets are immutable. However, SIM aims to maintain k *users* from a sequence of *actions* and each arrival action may induce updates in existing users' influence sets.

To bridge the gap between the two stream models and leverage existing algorithms based on the set-stream model, we propose a generic Set-Stream Mapping (SSM) interface. The interface makes two adaptations for a set-stream algorithm to serve as the checkpoint oracle. First, the candidate solution CX is adapted to store k users. Second, the objective function f' is adapted to the influence function $f(I_t[i](\cdot))$, where $I_t[i]$ denotes the influence set of user(s) over contiguous actions $\{W_t[i], \ldots, W_t[N]\}$. Subsequently, SSM maps an action stream to a set-stream and feeds the set-stream to $\Lambda_t[i]$. Whenever a new action a_t arrives, the following steps are taken for each $\Lambda_t[i]$:
(1) Identify users $u_1, u_2, \ldots, u_d$ whose $I_t[i](\cdot)$ is updated.
(2) Feed $\Lambda_t[i]$ with a stream $S'_t = \{I_t[i](u_1), \ldots, I_t[i](u_d)\}$.
(3) Update the solution of $\Lambda_t[i]$ for each $I_t[i](u) \in S'_t$.

There are several choices of oracles that are developed for the set-stream model with differences on the solution quality, update performance, and function generality. Typical oracles are listed in Table 2. An important conclusion is that our SSM procedure does not affect the quality guarantee of the mapped algorithms. Formally:

THEOREM 1. *Let $\mathcal{A}$ be an ε-approximate SSO algorithm in the set-stream model and $\overline{\mathcal{A}}$ be the mapped algorithm of $\mathcal{A}$ using SSM. Then, $\overline{\mathcal{A}}$ is also ε-approximate for SIM.*

PROOF. To show $\overline{\mathcal{A}}$ is also ε-approximate, we consider an append-only set-stream generated by SSM over the action stream. At any time t, let OPT_t be the optimal influence value of SIM wrt. W_t, and OPT^*_t be the optimal influence value achieved by any set of at most k sets from the mapped set-stream. We treat all influence sets in the mapped stream as independent sets regardless of whether they belong to the same user. We run $\overline{\mathcal{A}}$ on the mapped set-stream till time t and produce a result with at most k sets: $CX_t = \{I_{t_a}(u_{t_a}), \ldots, I_{t_b}(u_{t_b})\}$. Note that the influential sets in CX_t may be outdated and refer to the same user. Nevertheless, we can still use CX_t to approximate OPT_t without affecting the approximation ratio. To obtain the seed set from CX_t, we select a set of distinct users U_t from CX_t. Since the influence function f is monotone and the up-to-date influence set of any user always grows larger in the append-only stream, we have $f(I_t(U_t)) \geq f(\cup_{X \in CX_t} X)$. Moreover, CX_t is an ε-approximate solution over the append-only stream, i.e., $f(\cup_{X \in CX_t} X) \geq \varepsilon \mathsf{OPT}^*_t$. As the up-to-date influence sets always appear in the append-only set-stream, we have $\mathsf{OPT}^*_t \geq \mathsf{OPT}_t$, and thus $f(I_t(U_t)) \geq \varepsilon \mathsf{OPT}_t$. Therefore, U_t is an ε-approximate solution for SIM wrt. W_t. □

Table 2: Candidate checkpoint oracles

Oracle	Quality	Update	Function
SIEVESTREAMING [4]	$1/2 - \beta$	$O(\frac{\log k}{\beta})$	General
THRESHOLDSTREAM [19]	$1/2 - \beta$	$O(\frac{\log k}{\beta})$	General
Blog Watch [31]	1/4	$O(k)$	Cardinality
M*k*C [3]	1/4	$O(k \log k)$	Cardinality

According to the *SSM* steps, an action a_t is mapped to at most d influence sets, where d is the number of ancestors of a_t in its propagation. In practice, d is usually small, e.g., d is less than 5 on average as shown in our experiments (see Table 3). Since the number of checkpoints in the IC framework is N, the total number of checkpoint evaluations is $O(dN)$. If the update complexity of the checkpoint oracle for each set is $O(g)$, the total time complexity of the IC framework for each action is $O(dgN)$.

In the remaining of this section, we conduct a case study on using SIEVESTREAMING [4] as the checkpoint oracle. The adoption of other oracles can be similarly inferred.

4.3 A Case Study on SieveStreaming

The SieveStreaming Algorithm: SIEVESTREAMING [4] works as follows: Given a monotone submodular function f' and the optimal value OPT' of f' over the entire stream under a cardinality constraint, SIEVESTREAMING maintains a candidate solution CX that includes an incoming set X_t if CX has less than k sets and the following holds:

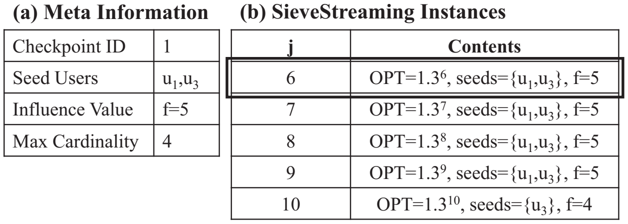
$$f'(CX \cup \{X_t\}) - f'(CX) \geq \frac{\frac{\mathsf{OPT}'}{2} - f'(CX)}{k - |CX|}$$

However, since OPT' is unknown in advance, SIEVESTREAMING maintains a sequence of possible values for OPT', i.e., $\Omega = \{(1+\beta)^j | j \in \mathbb{Z}, m \leq (1+\beta)^j \leq 2 \cdot k \cdot m\}$ where $m = \max_X f(\{X\})$ that has been observed. Accordingly, SIEVESTREAMING keeps $|\Omega| = O(\frac{\log k}{\beta})$ instances to ensure at least one of them achieves a $(1/2 - \beta)$ approximation ratio (see [4] for details).

Set-Stream Mapping for SieveStreaming: Following SSM, we create $\Lambda_t[i]$ as follows: Let CX_i be the user set maintained by $\Lambda_t[i]$. For each user u with her updated influence set $I_t[i](u)$, $\Lambda_t[i]$ selects u to CX_i if $|CX_i| < k$ and:

$$f(I_t[i](CX_i \cup \{u\})) - f(I_t[i](CX_i)) \geq \frac{\frac{\mathsf{OPT}_t[i]}{2} - f(I_t[i](CX_i))}{k - |CX_i|}$$

where $\mathsf{OPT}_t[i]$ is the optimal influence value achievable on all actions from $W_t[i]$ to $W_t[N]$. Similar to SIEVESTREAMING, each $\Lambda_t[i]$ keeps a set of possible values for $\mathsf{OPT}_t[i]$, i.e., $\Omega_t[i] = \{(1+\beta)^j | j \in \mathbb{Z}, m \leq (1+\beta)^j \leq 2 \cdot k \cdot m\}$



(a) Meta Information

Checkpoint ID	1
Seed Users	u_1,u_3
Influence Value	f=5
Max Cardinality	4

(b) SieveStreaming Instances

j	Contents
6	OPT=1.3^6, seeds={u_1,u_3}, f=5
7	OPT=1.3^7, seeds={u_1,u_3}, f=5
8	OPT=1.3^8, seeds={u_1,u_3}, f=5
9	OPT=1.3^9, seeds={u_1,u_3}, f=5
10	OPT=1.3^{10}, seeds={u_3}, f=4

Figure 3: Contents of Checkpoint $\Lambda_8[1]$.

where m denotes the maximum influence value of a single influence set over the actions $\{W_t[i], \ldots, W_t[N]\}$, i.e., $m = \max_{u \in U} f(I_t[i](u))$. And $|\Omega_t[i]| = O(\frac{\log k}{\beta})$ corresponding instances are maintained. To answer the SIM query, we always maintain the candidate user set achieving the largest influence value within the checkpoint. Figure 3 illustrates the content of a checkpoint when SieveStreaming is used as the checkpoint oracle.

Example 4. Figure 3 illustrates the contents of $\Lambda_8[1]$ in Example 3. $\Lambda_8[1]$ consists of the *meta information* and a sequence of SieveStreaming instances. In the *meta information*, the *Checkpoint ID* indicates the relative position of this checkpoint in the current window. The *Seed Users* and the *Influence Value* are maintained for query processing and checkpoint maintenance (as shown in Figure 3). The *Max Cardinality* is the maximum cardinality of a single user's influence set, i.e., $|I_8(u_3)| = 4$ for $\Lambda_8[1]$. Suppose $\beta = 0.3$, 5 candidates with $j = 6, \ldots, 10$ are maintained for $\Lambda_8[1]$ $(4 < 1.3^6 < \ldots < 1.3^{10} < 16)$. Each instance is maintained independently over the mapped set-stream and the instance with the largest influence value is used as the candidate solution (i.e., Instance with $j = 6$ highlighted in Figure 3).

Combining the results of Table 2 with Theorem 1, we can see that at least one user set maintained by $\Lambda_t[1]$ guarantees a $(1/2 - \beta)$-approximate solution for SIM wrt. W_t when SieveStreaming is used as the checkpoint oracle. In addition, the time complexity of IC for each update is $O(\frac{dN \log k}{\beta})$, since the update complexity of SieveStreaming is $O(\frac{\log k}{\beta})$.

5. SPARSE INFLUENTIAL CHECKPOINTS

In the IC framework, N checkpoints should be maintained to guarantee an ε-approximation ratio. This implies that $O(dN)$ checkpoint oracle updates need to be performed for each arrival action. However, real world applications often require millions of actions in one window. Therefore, it incurs prohibitive cost to maintain all checkpoints in practice.

To reduce the number of checkpoints maintained and thus improve the update efficiency, we design a *Sparse Influential Checkpoints* (SIC) framework to selectively maintain a subset of checkpoints without losing too much solution accuracy as the window shifts. Specifically, the number of checkpoints maintained by SIC is logarithmic with the window size N while its approximation ratio remains $\frac{\varepsilon(1-\beta)}{2}$ for any $\beta > 0$ if the checkpoint oracle is ε-approximate.

In this section, we first present the SIC framework and demonstrate its checkpoint maintenance in Section 5.1. In Section 5.2, we analyze the theoretical soundness and the complexity of SIC. Finally, we discuss how to generalize IC and SIC to handle multiple window shifts in Section 5.3.

5.1 The SIC Framework

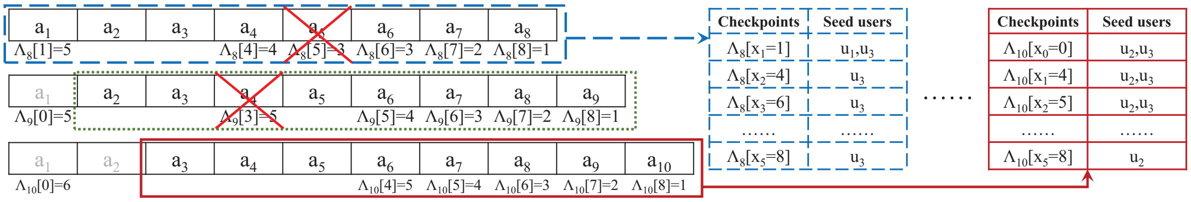
The idea of SIC is to leverage a subset of checkpoints to approximate the rest. On the one hand, to reduce the update cost, the number of checkpoints maintained should be as small as possible; on the other hand, the approximation ratio should remain tight. To achieve both goals, we propose a strategy to safely remove some checkpoints in the current window while ensuring the remaining checkpoints are able to approximate any windows with a bounded ratio.

We consider a sequence of checkpoints $\{\Lambda_t[x_0], \Lambda_t[x_1], \ldots, \Lambda_t[x_s]\}$ maintained by SIC at time t. Intuitively, given any three consecutive checkpoints $\Lambda_t[x_{i-1}]$, $\Lambda_t[x_i]$, $\Lambda_t[x_{i+1}]$ kept by SIC and a parameter $\beta \in (0,1)$, as long as $(1-\beta)\Lambda_t[x_{i-1}]$ is less than $\Lambda_t[x_i]$ and $\Lambda_t[x_{i+1}]$, we can safely delete $\Lambda_t[x_i]$ as $\Lambda_t[x_{i+1}]$ is at least $(1-\beta)$-approximate to $\Lambda_t[x_i]$. Given a checkpoint oracle with an ε-approximation for SIM, it is not hard to identify that using $\Lambda_t[x_{i+1}]$ for $\mathsf{OPT}_t[x_i]$ offers an $\varepsilon(1-\beta)$ approximate solution. Although such a maintenance strategy is simple, we need to ensure that the approximation ratio does not degrade seriously over time, i.e., the ratio should be at least $\frac{\varepsilon(1-\beta)}{2}$ at any time $t' > t$. We leave this rather complex analysis to Section 5.2 and focus on describing the maintenance procedure in the remaining of this subsection for ease of presentation.

Algorithm 2 SIC Maintenance

```
Require: SIC:$\{\Lambda_{t-1}[x_0], \Lambda_{t-1}[x_1], ..., \Lambda_{t-1}[x_s]\}$
 1: — on receiving action $a_t$ —
 2: Create $\Lambda_t[x_{s+1}]$ where $x_{s+1} = N$;
 3: for all $\Lambda_{t-1}[x_i]$ do
 4:     $\Lambda_t[x_i] \leftarrow \Lambda_{t-1}[x_i]$, $x_i \leftarrow x_i - 1$;
 5: for all $\Lambda_t[x_i]$ do
 6:     $\Lambda_t[x_i].process(a_t)$;
 7: for all $x_i$ do
 8:     $\Lambda^- \leftarrow \emptyset$;
 9:     for all $x_j > x_i$ do
10:         if $x_{j+1} \leq x_s$ and $\Lambda_t[x_j] \geq (1-\beta)\Lambda_t[x_i]$ and $\Lambda_t[x_{j+1}] \geq$
            $(1-\beta)\Lambda_t[x_i]$ then
11:             $\Lambda^- \leftarrow \Lambda^- \cup \{\Lambda_t[x_j]\}$;
12:         else
13:             break;
14:     Delete all checkpoints in $\Lambda^-$ from SIC;
15:     Shift the remaining checkpoints accordingly;
16: if $x_1 = 0$ then
17:     Delete $\Lambda_t[x_0]$ and shift the remaining checkpoints;
18: — on query —
19: return the solution of $\Lambda_t[x_1]$;
```

Algorithm 2 presents how to efficiently maintain the checkpoints over sliding windows in the SIC framework. Similar to the maintenance of IC, upon receiving a new action a_t, we create a new checkpoint for a_t (Line 2), add all checkpoints in W_{t-1} to W_t, and use a_t to update all checkpoints in W_t (Lines 3-6). Then the efficient deletion of checkpoints are presented in Lines 7-15. For each checkpoint $\Lambda_t[x_i]$, we find the first x_j $(j \geq i)$ such that $\Lambda_t[x_j] \geq (1-\beta)\Lambda_t[x_i]$ and $\Lambda_t[x_{j+1}] < (1-\beta)\Lambda_t[x_i]$. Then, all checkpoints between x_i and x_j are deleted and will be approximated by $\Lambda_t[x_j]$ in the subsequent window shifts. Finally, if the second checkpoint (i.e., $\Lambda_t[x_1]$) has expired, the earliest checkpoint (i.e., $\Lambda_t[x_0]$) will be deleted (Lines 16-17). It is notable that an additional checkpoint ($\Lambda_t[x_0]$) is stored in SIC to keep track of the solution over a window with size larger than N. Since $\Lambda_t[x_0]$ approximates the upper bound of the optimal solution for the current window and Algorithm 2 always main-



Checkpoints	Seed users
$\Lambda_8[x_1=1]$	u_1,u_3
$\Lambda_8[x_2=4]$	u_3
$\Lambda_8[x_3=6]$	u_3
……	……
$\Lambda_8[x_5=8]$	u_3

Checkpoints	Seed users
$\Lambda_{10}[x_0=0]$	u_2,u_3
$\Lambda_{10}[x_1=4]$	u_2,u_3
$\Lambda_{10}[x_2=5]$	u_2,u_3
……	……
$\Lambda_{10}[x_5=8]$	u_2

Figure 4: An example of checkpoint maintenance in SIC.

tains a bounded ratio between two neighboring checkpoints, a bounded approximation ratio is guaranteed by using $\Lambda_t[x_1]$ as the solution for the current window. Figure 4 and Example 5 provide the running example of the SIC maintenance.

Example 5. Figure 4 illustrates the maintenance of checkpoints in the SIC framework following Example 1. Let $N = 8$, $k = 2$ and $\beta = 0.3$. There are initially 6 checkpoints in SIC at time 8. According to Algorithm 2, $\Lambda_8[5]$ is deleted from SIC since $\Lambda_8[6] = 3 > (1 - 0.3) \times 3 = (1 - \beta)\Lambda_8[4]$. At time 8, $\Lambda_8[1]$ will be used to answer the SIM query. As the window shifts at time 9 with the arrival of a_9, a_1 and $\Lambda_8[1]$ (which later becomes $\Lambda_9[0]$) expire. But $\Lambda_9[0]$ is stored because $\Lambda_9[3]$ has not expired yet. Then all checkpoints will be updated according to a_9. After the update procedure, we find $\Lambda_9[3]$ can be deleted since $\Lambda_9[5] > (1-\beta)\Lambda_9[0]$. Finally, all checkpoints are updated according to a_{10} at time 10 and no checkpoint is to be deleted. $\Lambda_{10}[4]$ will be used to answer the SIM query at time 10.

In the following, we will demonstrate the theoretical soundness of SIC and also analyze the complexity of SIC.

5.2 Theoretical Analysis

To establish our theoretical claims for SIC, we first analyze the property of the optimal checkpoint oracle which always returns the optimal solution for SIM over an append-only action stream. There are two important properties of the optimal checkpoint oracle.

Definition 3 (Monotonicity & Subadditivity). *Let $t_a \leq t_b$ be two timestamps and $W_{t_b}^{t_a}$ represents a window containing a set of contiguous actions: $a_{t_a}, ..., a_{t_b}$ with the corresponding checkpoint denoted as $\Lambda_{t_b}^{t_a}$. Given any t_1, t_2, t_3 s.t. $t_1 \leq t_2 \leq t_3$, the checkpoint is monotone if $\Lambda_{t_3}^{t_1} \geq \Lambda_{t_2}^{t_1}$. Moreover, the checkpoint is subadditive if $\Lambda_{t_3}^{t_1} \leq \Lambda_{t_2}^{t_1} + \Lambda_{t_3}^{t_2}$.*

Lemma 1. *Let $t_a \leq t_b$ be two timestamps and $\mathsf{OPT}_{t_b}^{t_a}$ denote the optimal oracle (as well as the optimal value) for $W_{t_b}^{t_a}$. The optimal oracle is both monotone and subadditive.*

The proof of Lemma 1 can be found in [1]. We omit it here due to space limitations. We note that although the optimal checkpoint oracle is both monotone and subadditive, it is intractable unless $P = NP$. In practice, we utilize the approximate checkpoint oracles as listed in Table 2. The approximate oracles are monotone. This is essential due to their greedy nature: updating the maintained result only when this update increases the function value. Given the monotonicity of the approximate checkpoint oracles, the monotonicity and subadditivity of the optimal oracle, we are ready to prove that the checkpoint maintenance strategy used in SIC is theoretically bounded.

Lemma 2. *Given any t_1, t_2, t_3, t_4 s.t. $t_1 \leq t_2 \leq t_3 \leq t_4$, $\forall \beta \in (0, 1)$, if $(1 - \beta)\Lambda_{t_3}^{t_1} \leq \Lambda_{t_3}^{t_2}$, then $\frac{\varepsilon(1-\beta)}{2}\mathsf{OPT}_{t_4}^{t_1} \leq \Lambda_{t_4}^{t_2}$.*

Proof. The following inequalities hold:

$$\begin{aligned}\Lambda_{t_4}^{t_2} &\geq \frac{1}{2}(\Lambda_{t_3}^{t_2} + \Lambda_{t_4}^{t_2}) \geq \frac{1}{2}((1-\beta)\Lambda_{t_3}^{t_1} + \Lambda_{t_4}^{t_2}) \\ &\geq \frac{1-\beta}{2}(\Lambda_{t_3}^{t_1} + \Lambda_{t_4}^{t_2}) \geq \frac{\varepsilon(1-\beta)}{2}(\mathsf{OPT}_{t_3}^{t_1} + \mathsf{OPT}_{t_4}^{t_2}) \\ &\geq \frac{\varepsilon(1-\beta)}{2}\mathsf{OPT}_{t_4}^{t_1}\end{aligned}$$

where the first inequality holds from the monotonicity of the approximate checkpoint oracles; the second inequality is due to the condition that $\Lambda_{t_3}^{t_2} \geq (1-\beta)\Lambda_{t_3}^{t_1}$; the third inequality is obvious since $\beta \in (0, 1)$; the fourth inequality holds because of the approximation ratio of checkpoint oracles and the final inequality holds as the optimal checkpoint oracle is both monotone and subadditive. □

According to Lemma 2, if $(1 - \beta)\Lambda_{t_3}^{t_1} \leq \Lambda_{t_3}^{t_2}$, using the checkpoint oracle started at t_2 to approximate any checkpoints between t_1 and t_2 always achieves an $\frac{\varepsilon(1-\beta)}{2}$ approximation for any number of appending actions. Next, we present Lemma 3 to demonstrate the property of the checkpoints maintained by Algorithm 2.

Lemma 3. *The SIC on window W_t contains s checkpoints $\Lambda_t[x_0], \Lambda_t[x_1], \ldots, \Lambda_t[x_s]$ $(x_0 < x_1 < \ldots < x_s)$ maintained by Algorithm 2. Given a constant $\beta \in (0, 1)$, any neighboring checkpoints $\Lambda_t[x_i]$, $\Lambda_t[x_{i+1}]$ and $\Lambda_t[x_{i+2}]$ satisfy one of the following conditions:*

1. *if $\Lambda_t[x_{i+1}] \geq (1-\beta)\Lambda_t[x_i]$, then $\Lambda_t[x_{i+2}] < (1-\beta)\Lambda_t[x_i]$.*
2. *if $x_{i+1} \neq x_i + 1 \wedge \Lambda_t[x_{i+1}] < (1 - \beta)\Lambda_t[x_i]$, then $\frac{\varepsilon(1-\beta)}{2} \cdot \mathsf{OPT}_t[x_i] \leq \Lambda_t[x_{i+1}]$.*
3. *$x_{i+1} = x_i + 1 \wedge \Lambda_t[x_{i+1}] < (1 - \beta)\Lambda_t[x_i]$.*

Proof. We prove the lemma by induction. As the base case, there are only 2 actions in the window and either condition 1 or condition 3 holds.

Next, assuming Lemma 3 holds at time t and we show that it still holds after the update procedure in Algorithm 2 at time $t+1$. Let $\Lambda_t[x_i]$ be a checkpoint instantiated before $t+1$ and is not deleted during the update procedure at $t+1$, then $\Lambda_t[x_{i+1}]$ is the subsequent checkpoint of $\Lambda_{t+1}[x_i]$ at time t. Next, we discuss all possible cases when performing the update procedure of Algorithm 2 at time $t + 1$:
Case 1: $x_{i+1} \neq x_i+1$ and $\Lambda_t[x_{i+1}]$ is deleted at $t+1$. In this case, we have $\Lambda_{t+1}[x_{i+1}] \geq (1-\beta)\Lambda_{t+1}[x_i]$ and $\Lambda_{t+1}[x_{i+2}] < (1 - \beta)\Lambda_{t+1}[x_i]$ according to Lines 7-15 of Algorithm 2. In this case, condition 1 holds at $t + 1$.
Case 2: $x_{i+1} \neq x_i + 1$ and $\Lambda_{t+1}[x_{i+1}]$ is not deleted at $t + 1$. In this case, $\Lambda_{t+1}[x_{i+1}]$ must become the subsequent checkpoint of $\Lambda_{t+1}[x_i]$ at some time $t' \leq t$. Then, at t',

we have $\Lambda_{t'}[x_{i+1}] \geq (1-\beta)\Lambda_{t'}[x_i]$. According to Lemma 2, $\Lambda_{t+1}[x_{i+1}] \geq \frac{\varepsilon(1-\beta)}{2}\mathsf{OPT}_{t+1}[x_i]$ holds. Because $\Lambda_{t+1}[x_{i+1}]$ is not deleted at $t+1$, we have either condition 1 (when $\Lambda_{t+1}[x_{i+1}] \geq (1-\beta)\Lambda_{t+1}[x_i]$) or condition 2 holds (when $\Lambda_{t+1}[x_{i+1}] < (1-\beta)\Lambda_{t+1}[x_i]$) at $t+1$.

Case 3: $x_{i+1} = x_i + 1$. If $\Lambda_{t+1}[x_{i+1}] \geq (1-\beta)\Lambda_{t+1}[x_i]$, then condition 1 holds since $\Lambda_{t+1}[x_{i+1}]$ is not deleted at $t+1$; otherwise, condition 3 holds.

Therefore, at least one condition in Lemma 3 holds in all possible cases at $t+1$ and we conclude the proof. □

Leveraging Lemma 3, we are able to analyze SIC theoretically. First, from conditions 1 and 2, we can infer that if there are checkpoints deleted between x_i and x_{i+1}, the ratios between $\Lambda_t[x_{i+1}]$ and the optimal solution of deleted checkpoints are guaranteed to be at least $\frac{\varepsilon(1-\beta)}{2}$. Next, by collectively examining conditions 1–3, we can see that there is at least one checkpoint in $\Lambda_t[x_{i+1}]$ and $\Lambda_t[x_{i+2}]$ returning an influence value of smaller than $(1-\beta)\Lambda_t[x_i]$, and thus the number of checkpoints maintained is $O(\frac{\log N}{\beta})$. Based on these intuitions, we then formally state the approximation guarantee and the complexity of SIC in Theorems 2–4:

THEOREM 2. *SIC maintains a* $\frac{\varepsilon(1-\beta)}{2}$*-approximate solution for SIM in* $\Lambda_t[x_1]$ *when an* ε*-approximate checkpoint oracle is used.*

PROOF. We use OPT_t to denote the optimal solution of SIM w.r.t. W_t and we prove that $\frac{\varepsilon(1-\beta)}{2}$ is a lower bound for the ratio between $\Lambda_t[x_1]$ and OPT_t.

Let $\Lambda_t[x_0]$ be the expired checkpoint just before $\Lambda_t[x_1]$. Since $\Lambda_t[x_0]$ and $\Lambda_t[x_1]$ are neighboring checkpoints in SIC, one of the conditions in Lemma 3 holds at time t.

If condition 3 in Lemma 3 holds, we have $\mathsf{OPT}_t \leq \varepsilon\Lambda_t[x_1]$ since $\Lambda_t[x_1]$ directly maintains an approximate solution on W_t. Otherwise, we have: $\mathsf{OPT}_t \leq \mathsf{OPT}_t[x_0] \leq \frac{2}{\varepsilon(1-\beta)}\Lambda_t[x_1]$ since $\Lambda_t[x_0]$ has expired. Thus, SIC maintains an at least $\frac{\varepsilon(1-\beta)}{2}$-approximate solution in $\Lambda_t[x_1]$. □

THEOREM 3. *SIC obtains a* $(1/4-\beta)$*-approximate solution for SIM when* SIEVESTREAMING *is used as the checkpoint oracle.*

PROOF. Since the SIEVESTREAMING algorithm guarantees a $(1/2-\beta)$ approximation ratio to the optimal solution, SIC with SIEVESTREAMING as the checkpoint oracle preserves a $\frac{1}{2}(\frac{1}{2}-\beta)(1-\beta)$ approximation guarantee according to Theorem 2. As $\frac{1}{2}(\frac{1}{2}-\beta)(1-\beta) = \frac{1}{4} - \frac{3}{4}\beta + \beta^2 > \frac{1}{4} - \beta$, we get at least a $(1/4-\beta)$-approximate solution for SIM. □

THEOREM 4. *The number of checkpoints maintained by SIC wrt. a sliding window of size* N *is* $O(\frac{\log N}{\beta})$.

PROOF. Lemma 3 guarantees either $\Lambda_t[x_{i+1}]$ or $\Lambda_t[x_{i+2}]$ is less than $(1-\beta)\Lambda_t[x_i]$. Since $\Lambda_t[x_1]/\Lambda_t[N]$ is bounded by $O(N)$, the number of checkpoints is at most $\frac{2\cdot\log N}{\log(1-\beta)^{-1}}$ for $\beta \in (0,1)$. Therefore, the number of checkpoints maintained by SIC is $O(\frac{\log N}{\beta})$. □

As the time complexity for a checkpoint to update each action is $O(dg)$ if each checkpoint takes $O(g)$ to evaluate one influence set and the number of checkpoints maintained by SIC is $O(\frac{\log N}{\beta})$, the time complexity of SIC to update each action is $O(\frac{dg\log N}{\beta})$. When SIEVESTREAMING is used as the checkpoint oracle, we have $g = O(\frac{\log k}{\beta})$ and thus the time complexity of SIC for each update is $O(\frac{d\log N\log k}{\beta^2})$.

5.3 Handling Multiple Window Shifts

Although we have discussed how to handle SIM queries for windows which shift for one action at a time, many applications do not require to retrieve the result at such an intense rate. Hereby, we discuss how to handle multiple window shifts, i.e., each window shift receives L new actions while the earliest L actions become expired at the same time.

To handle multiple window shifts for IC, we create only one new checkpoint and delete the earliest checkpoint when the window shifts from W_t to W_{t+L}. Subsequently, all actions from a_{t+1} to a_{t+L} are collected to update all checkpoints in the window. Thus, the number of checkpoints created for multiple window shifts will be $\lceil\frac{N}{L}\rceil$. On top of the IC maintenance strategy, we still use the same SIC algorithm over the checkpoints created by IC to support multiple window shifts.

Lastly, the aforementioned maintenance strategies still preserve the theoretical results as there is no fundamental differences between handling single window shift and multiple window shifts using our proposed frameworks.

6. EXPERIMENTAL RESULTS

In this section, we evaluate the efficiency and effectiveness of our proposed frameworks on several real-world and synthetic datasets. First, we compare IC and SIC for influence values and processing efficiency with varying β. Then, we compare the solution qualities and throughputs of all approaches with different seed set sizes. Finally, we evaluate the scalability of all compared approaches.

6.1 Experimental Setup

Datasets: We collect two real-world datasets and synthesize two datasets for extensive studies.

- **Reddit**: Reddit is an online forum where user actions include *post* and *comment*. We collect all Reddit *comment* actions in May 2015 from *kaggle*[6] and query the Reddit API for the *post* actions in the same period. The dataset contains 48,104,875 actions from 2,628,904 users.
- **Twitter**: Twitter is an online social network where actions include *tweet*, *retweet*, *quote* and *reply*. We crawl these actions for one week via Twitter stream API[7] on trending topics such as US presidential election, 2016 NBA finals and Euro 2016. The dataset contains 9,724,908 actions from 2,881,154 users.
- **Synthetic Datasets**: We synthesize two action streams with different response patterns to test the robustness of the proposed solutions. There are two types of actions in concern: *post* and *follow*. We use the R-MAT model [9] to synthesize 5 different power law graphs with the number of users ranging from 1-5 million (2 million by default). For each synthetic graph, we generate 10 million actions by randomly selecting a user to perform either a *post* or a *follow* action. If an action a_t is *follow*, it will respond to a previous action $a_{t'}$ with a response distance $\Delta = t - t'$. To demonstrate different response patterns, two datasets are generated based on the distances conforming to exponential distributions with different parameters: (1)**SYN-O**: $\Delta \sim \mathtt{exp}(\lambda = 2.0 \times 10^{-6})$, which indicates "old posts get

[6] `https://www.kaggle.com/reddit/reddit-comments-may-2015`

[7] `https://dev.twitter.com/streaming/overview`

Table 3: Statistics on datasets

Dataset	Users	Actions	Resp. dist.	Avg. depth
Reddit	2,628,904	48,104,875	404,715	4.58
Twitter	2,881,154	9,724,908	294,609	1.87
SYN-O	1M–5M	10,000,000	500,000	2.5
SYN-N	1M–5M	10,000,000	5,000	2.59

more followers"; (2)**SYN-N**: $\Delta \sim \texttt{exp}(\lambda = 2.0 \times 10^{-4})$, which represents "recent posts get more followers".

The statistics of these datasets are summarized in Table 3.

Approaches: All approaches compared in the experiments are listed as follows:

- **IMM** [35]: To support our argument on the effectiveness, we use the state-of-the-art IM algorithm on static graphs as a baseline. At each time t, we construct an influence graph G_t by treating users as vertices and the *influence* relationships between users wrt. W_t as directed edges. The edge probabilities between users are assigned by the weighted cascade (WC) [18] model. To extract the influential users at time t, we set the parameters of IMM to be $\varepsilon = 0.5$, $l = 1$ [35] and run the algorithm on the generated influence graph G_t.
- **UBI** [11]: We use the state-of-the-art method for IM on dynamic graphs as another baseline. The generation of influence graphs is the same as *IMM*. Then, a sequence of influence graphs $\{G_1, \ldots, G_m\}$ are fed to *UBI* in a chronological order to track the influential users. We keep the same interchange threshold as used in [11], i.e., $\gamma = 0.01$.
- **Greedy** [28]: We also implement the classic greedy algorithm in [28] since it achieves the best theoretical approximation (i.e., $1-1/e$) of SIM queries. A detailed description of this algorithm is presented in Section 4. Since the *Greedy* algorithm does not store any intermediate result, it always recomputes the solution when being queried.
- **IC**: The IC framework proposed in Section 4. We use SieveStreaming [4] as the checkpoint oracle.
- **SIC**: The SIC framework proposed in Section 5. We use the same checkpoint oracle as IC.

Quality Metric: We note that *IMM* and *UBI* work under the WC model whereas *Greedy*, *IC* and *SIC* are proposed to answer SIM queries in Section 3. To verify the effectiveness of our proposed solutions, we retrieve the seed users returned by all approaches for each window shift. When a set of seed users is returned by each approach at time t, we evaluate the *influence spread* of the users under the WC model with 10,000 rounds of Monte-Carlo simulation on the corresponding influence graph G_t. Finally, we use the *average influence spread* of all windows for each approach as the quality metric.

Performance Metric: We use *throughput* as our performance metric. Specifically, whenever the window shifts for L actions, we measure the elapsed CPU time of each approach and the throughput is L divided by the elapsed time. We do not measure the query processing time because all approaches maintain the seed users explicitly and the time to retrieve them is negligible.

Parameters: The parameters examined in our experiments: (1) β is the parameter in *IC* and *SIC* to achieve a trade-off between quality and efficiency. (2) k is the size of the seed set. (3) N is the window size. (4) L is the number of actions for each window shift. (5) $|U|$ is the total number of users for synthetic datasets. We vary N, L and $|U|$ to test the scalability of the compared approaches. The summary of parameters is listed in Table 4. The default values of all parameters are in bold.

Table 4: Parameters in experiments

Parameter	Values
k	5, 25, **50**, 75, 100
β	0.1, **0.2**, 0.3, 0.4, 0.5
N	100K, 250K, **500K**, 750K, 1,000K
L	1K, 2.5K, **5K**, 7.5K, 10K
$\lvert U\rvert$	1M, **2M**, 3M, 4M, 5M

Experiment Settings: All experiments are conducted on a desktop machine running Ubuntu 14.04 with a quad core 3.4 GHz Intel i7-2600 processor and 16 GB memory. All the approaches except *IMM* are implemented in Java 8. The *IMM* implementation available[8] is written in C++.

6.2 Testing β for IC and SIC

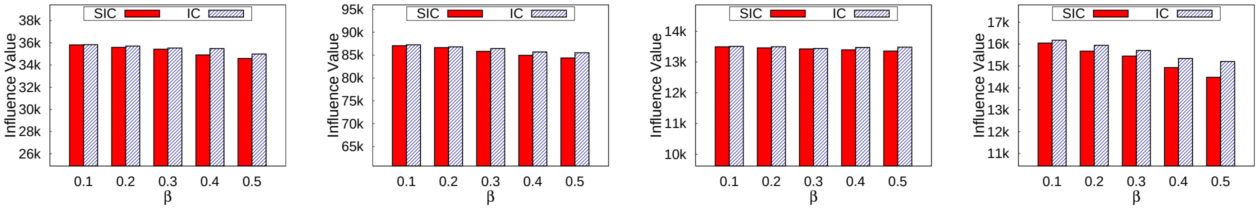
We first vary β to test its effect on *IC* and *SIC* in terms of the average influence value of SIM queries using the cardinality function, the number of maintained checkpoints and the throughput. Note that we compare the seed qualities of *IC* and *SIC* with the baselines in Section 6.3 and only focus on their effectiveness of answering SIM queries here.

Influence Value: The influence values of *IC* and *SIC* with varying β are presented in Figure 5a–5d. The influence values of *IC* are slightly better than *SIC* in most experiments. This is because *SIC* trades quality for efficiency by maintaining fewer checkpoints. In spite of that, *SIC* is able to obtain competitive values with at most 5% off from *IC*. In addition, we can see that both *SIC* and *IC* achieve better influence values for a smaller β and the influence values of *SIC* degrade faster than *IC* for a larger β due to the deletion of checkpoints. We note that in the SYN-N dataset, the influence values of *SIC* degrade more severely than other datasets for a larger β. This is because the average reply distance is very short, which leads to the frequent changes of the influential users. Nevertheless, *SIC* still returns solutions within the theoretical bound stated in Section 5.2.

Number of checkpoints: We examine the average number of checkpoints maintained by *IC* and *SIC* for all sliding windows. The results are presented in Figure 6a–6d. *IC* maintains a constant number of checkpoints wrt. β. This is because the number of checkpoints maintained by *IC* in each sliding window exactly equals to $\lceil \frac{N}{L} \rceil$. On the contrary, the number of checkpoints in *SIC* is $O(\frac{\log N}{\beta})$ according to Theorem 4 in Section 5.2, and is thus negatively correlated with β. The trend for the number of checkpoints emphasizes the superiority of *SIC* in both space and time efficiencies.

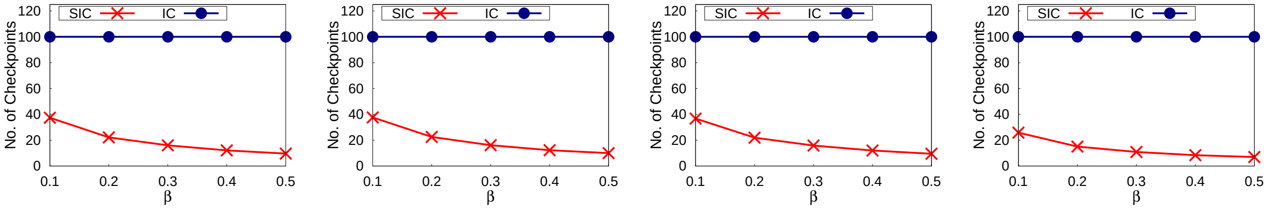
Throughput: The throughputs of *IC* and *SIC* are presented in Figure 7a–7d. Both *IC* and *SIC* achieve better performance for a larger β. There are two reasons behind such an observation. First, both approaches employ SieveStreaming as the checkpoint oracle where fewer candidate instances are kept within each checkpoint for a larger β, which makes the update time shorter for each checkpoint. Second, *SIC* maintains fewer checkpoints as β becomes larger, which naturally leads to shorter processing time. Thus, for a larger β, *SIC* shows even more superior-

[8] `https://sourceforge.net/projects/im-imm/`



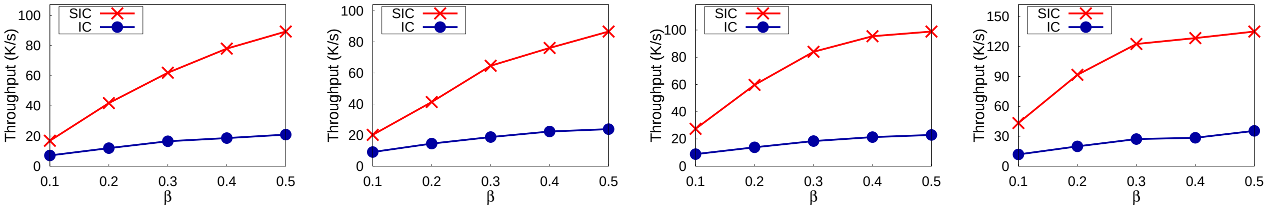
(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 5: Influence Values of IC and SIC with varying β.



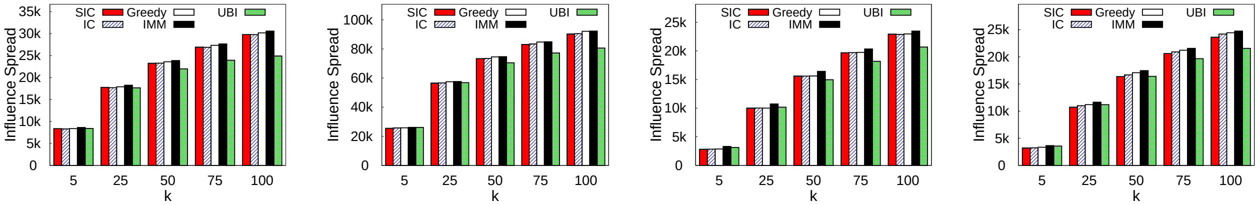
(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 6: The number of checkpoints maintained by IC and SIC with varying β.



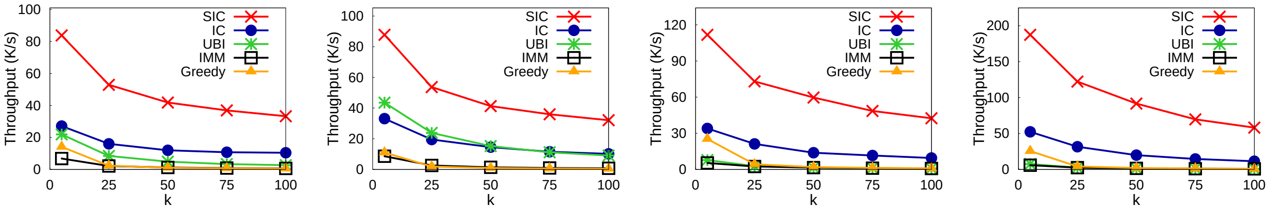
(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 7: Throughputs of IC and SIC with varying β.



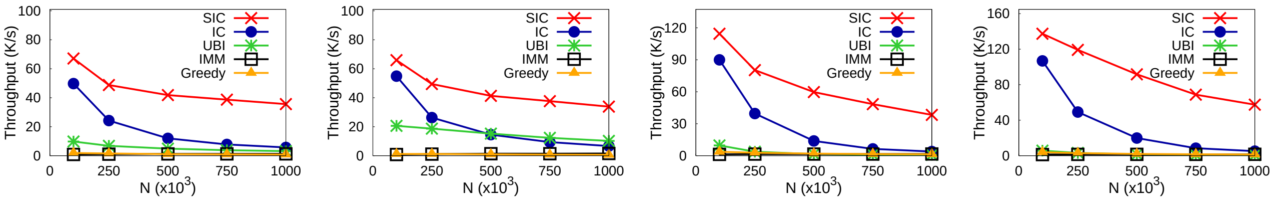
(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 8: Solution qualities of compared methods with varying k.



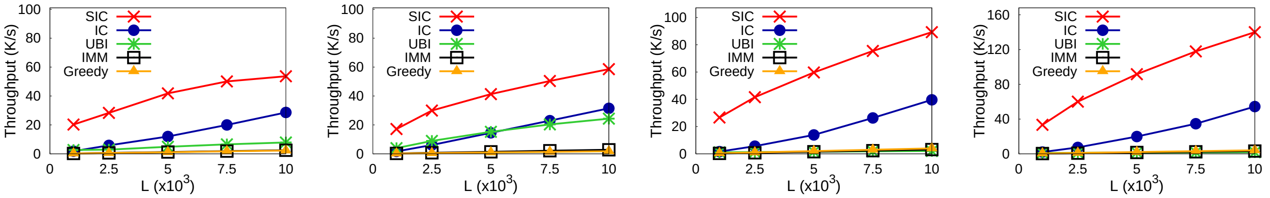
(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 9: Throughputs of compared methods with varying k.



(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 10: Throughputs of compared approaches with varying W.



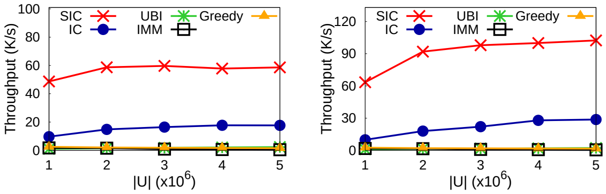
(a) Reddit (b) Twitter (c) SYN-O (d) SYN-N

Figure 11: Throughputs of compared approaches with varying L.

ity over *IC* in all experiments because fewer checkpoints are maintained by *SIC*.

6.3 Comparing Approaches with Varying k

We compare different approaches by using the quality and the performance metric defined in Section 6.1 for varying k.
Quality: The results of solution qualities for different approaches are presented in Figure 8a–8d. Compared with *IMM*, *Greedy*, *IC* and *SIC* achieve less than 10% quality losses. The results have verified the effectiveness of SIM as the seeds for SIM queries achieve nearly equivalent influence spreads as the seeds retrieved by IMM under the WC model. Moreover, *SIC* shows competitive qualities though it maintains fewer checkpoints than *IC*. In contrast, the qualities of *UBI* are close to *IMM* when k is small (i.e., $k \leq 25$). But its qualities degrade dramatically when k increases. This is because *UBI* relies on interchanging users to maintain the influential users against the updates of the influence graph. It interchanges a user into the maintained influential user set only when a substantial gain is achieved for the estimated influence spread (i.e., 1% of the total influence spread prior to the interchange). Thus, for a larger k, it becomes harder for a user to be interchanged since the total influence spread of the maintained user set is larger. This results in the delays of interchanges and causes larger errors.
Throughput: The performances with varying k are presented in Figure 9a–9d. The throughputs of all approaches are inversely proportional to k. *IC* and *SIC* both employ SieveStreaming as the checkpoint oracle, each checkpoint maintains a number of candidate instances and each instance contains up to k candidate users. When k gets larger, it is more expensive to evaluate the influence function for each checkpoint. This explains why the performances of *IC* and *SIC* drop while k becomes larger. Compared with *IC* and all baseline methods, *SIC* shows significant advantages in efficiency for all experiments. Moreover, *SIC* dominates *Greedy* and *IMM* by achieving a speedup of up to 2 orders of magnitude across all datasets. The throughput of *UBI* is also far behind *SIC* and *IC* on all datasets except *Twitter*, where it achieves an equivalent throughput compared to *IC*. Nonetheless, *UBI* is still more than 3 times slower than *SIC*.



(a) SYN-O (b) SYN-N

Figure 12: Throughputs with varying $|U|$.

6.4 Scalability

In this section, we evaluate the scalability of compared methods through measuring the throughputs when varying the window size N, the length of each window shift L and the total number of users $|U|$.
Varying N: The performances for varying N are presented in Figure 10a–10d. Although the throughputs of all approaches decrease with increasing N, *SIC* shows better scalability as it only maintains $O(\log N)$ checkpoints when β is fixed. We observe a smaller performance gap between *IC* and *SIC* in all datasets when N is small (i.e., $N = 100,000$). This is because the number of checkpoints maintained by *IC* are very close to *SIC* (fewer than 8) and the benefits of sparse checkpoints become less significant. Nonetheless, when N increases, *SIC* regains its superiority. Moreover, *SIC* achieves speedups of up to 40x, 100x and 70x compared to *Greedy*, *IMM* and *UBI* respectively.
Varying L: We show the performances for varying L in Figure 11a–11d. As L increases, the throughputs of *IC* and *SIC* increase. This is because larger L results in a smaller number of checkpoints for both methods. *IC* exhibits a linear performance improvement wrt. larger L since it main-

tains $\lceil \frac{N}{L} \rceil$ checkpoints. As *SIC* deletes some checkpoints created by *IC*, it continues to be superior to *IC* in terms of performance, which demonstrates its scalability in handling multiple window shifts. Like the results for varying N, *SIC* dominates *Greedy*, *IMM* and *UBI* in terms of the throughput.

Varying $|U|$: Finally, we show the performances for varying $|U|$ on two synthetic datasets in Figure 12a–12b. We observe that the throughputs of *SIC*, *IC* and *UBI* increase as $|U|$ becomes larger. Fixing the window size N, the influence graphs become more sparse for a larger $|U|$. All these three approaches show better performance on more sparse graphs. However, *Greedy* and *IMM* need more processing time to run as $|U|$ increases since these complexities are directly related to $|U|$. Finally, *SIC* still shows superior performances in all experiments with various $|U|$ settings.

7. CONCLUSION

In this paper, we proposed a novel *Stream Influence Maximization* (SIM) query to retrieve k influential users who collectively maximized the influence value over a social action stream. Then, we presented a novel framework *Influential Checkpoints* (IC) and its improved version *Sparse Influential Checkpoints* (SIC) to efficiently support the continuous SIM queries over high-speed social streams. Theoretically, SIC maintained $O(\frac{\log N}{\beta})$ checkpoints to obtain an $\frac{\epsilon(1-\beta)}{2}$-approximate solution for SIM queries. Empirically, our experiments showed that SIC achieved up to 2 orders of magnitude speedups over the state-of-the-art static and dynamic IM approaches with less than 10% losses in seed quality. In particular, SIC demonstrated a peak processing rate of more than 150K actions per second, which was adequate for real-world social streams. In the future, we plan to extend our proposed frameworks to support a broader class of IM problems, e.g., competitive IM [6, 23, 27].

8. REFERENCES

[1] https://arxiv.org/abs/1702.01586.

[2] C. C. Aggarwal, S. Lin, and P. S. Yu. On influential node discovery in dynamic social networks. In *SDM*, pages 636–647, 2012.

[3] G. Ausiello, N. Boria, A. Giannakos, G. Lucarelli, and V. T. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13–14):1901–1913, 2012.

[4] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, pages 671–680, 2014.

[5] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *ICDM*, pages 81–90, 2012.

[6] S. Bharathi, D. Kempe, and M. Salek. Competitive influence maximization in social networks. In *WINE*, pages 306–311, 2007.

[7] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, pages 946–957, 2014.

[8] V. Braverman and R. Ostrovsky. Smooth histograms for sliding windows. In *FOCS*, pages 283–293, 2007.

[9] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SDM*, pages 442–446, 2004.

[10] S. Chen, J. Fan, G. Li, J. Feng, K.-L. Tan, and J. Tang. Online topic-aware influence maximization. *PVLDB*, 8(6):666–677, 2015.

[11] X. Chen, G. Song, X. He, and K. Xie. On influential nodes tracking in dynamic social networks. In *SDM*, pages 613–621, 2015.

[12] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[13] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[14] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[15] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250, 2010.

[16] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1):73–84, 2011.

[17] L. Guo, D. Zhang, W. Wu, G. Cong, and K.-L. Tan. Influence maximization in trajectory databases. *IEEE Transactions on Knowledge and Data Engineering*, 29(3):627–641, 2017.

[18] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[19] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing*, 2(3):1–14, 2015.

[20] K. Kutzkov, A. Bifet, F. Bonchi, and A. Gionis. Strip: Stream learning of influence probabilities. In *KDD*, pages 275–283, 2013.

[21] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[22] G. Li, S. Chen, J. Feng, K.-L. Tan, and W.-S. Li. Efficient location-aware influence maximization. In *SIGMOD*, pages 87–98, 2014.

[23] H. Li, S. S. Bhowmick, J. Cui, Y. Gao, and J. Ma. Getreal: Towards realistic selection of influence maximization strategies in competitive networks. In *SIGMOD*, pages 1525–1537, 2015.

[24] H. Li, S. S. Bhowmick, A. Sun, and J. Cui. Conformity-aware influence maximization in online social networks. *The VLDB Journal*, 24(1):117–141, 2015.

[25] Y. Li, J. Fan, D. Zhang, and K.-L. Tan. Discovering your selling points: Personalized social influential tag exploration. In *SIGMOD*, 2017. to appear.

[26] Y. Li, D. Zhang, and K.-L. Tan. Real-time targeted influence maximization for online advertisements. *PVLDB*, 8(10):1070–1081, 2015.

[27] W. Lu, W. Chen, and L. V. S. Lakshmanan. From competition to complementarity: Comparative influence diffusion and maximization. *PVLDB*, 9(2):60–71, 2015.

[28] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[29] H. T. Nguyen, M. T. Thai, and T. N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *SIGMOD*, pages 695–710, 2016.

[30] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-i. Kawarabayashi. Dynamic influence analysis in evolving networks. *PVLDB*, 9(12):1077–1088, 2016.

[31] B. Saha and L. Getoor. On maximum coverage in the streaming model and application to multi-topic blog-watch. In *SDM*, pages 697–708, 2009.

[32] G. Soda, A. Usai, and A. Zaheer. Network memory: The influence of past and current networks on performance. *Academy of Management Journal*, 47(6):893–906, 2004.

[33] X. Song, B. L. Tseng, C.-Y. Lin, and M.-T. Sun. Personalized recommendation driven by information flow. In *SIGIR*, pages 509–516, 2006.

[34] K. Subbian, C. C. Aggarwal, and J. Srivastava. Querying and tracking influencers in social streams. In *WSDM*, pages 493–502, 2016.

[35] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *SIGMOD*, pages 1539–1554, 2015.

[36] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: near-optimal time complexity meets practical efficiency. In *SIGMOD*, pages 75–86, 2014.

[37] X. Wang, Y. Zhang, W. Zhang, and X. Lin. Distance-aware influence maximization in geo-social network. In *ICDE*, pages 1–12, 2016.

[38] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun. Influence maximization in dynamic social networks. In *ICDM*, pages 1313–1318, 2013.