4-2018

Location-aware influence maximization over dynamic social streams

Yanhao WANG
National University of Singapore

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

Ju FAN
Renmin University of China

Kianlee TAN
National University of Singapore

DOI: https://doi.org/10.1145/3230871

Citation

WANG, Yanhao; LI, Yuchen; FAN, Ju; and TAN, Kianlee. Location-aware influence maximization over dynamic social streams. (2018). *ACM Transactions on Information Systems*. 36, (4), 43: 1-35. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4155

Location-aware Influence Maximization over Dynamic Social Streams

YANHAO WANG, National University of Singapore, Singapore
YUCHEN LI, Singapore Management University, Singapore
JU FAN, Renmin University of China, China
KIAN-LEE TAN, National University of Singapore, Singapore

Influence maximization (IM), which selects a set of k seed users (a.k.a., *a seed set*) to maximize the influence spread over a social network, is a fundamental problem in a wide range of applications. However, most existing IM algorithms are static and location-unaware. They fail to provide high-quality seed sets efficiently when the social network evolves rapidly and IM queries are location-aware. In this article, we first define two IM queries, namely *Stream Influence Maximization* (SIM) and *Location-aware SIM* (LSIM), to track influential users over social streams. Technically, SIM adopts the sliding window model and maintains a seed set with the maximum influence value collectively over the most recent social actions. LSIM further considers social actions are associated with *geo-tags* and identifies a seed set that maximizes the influence value in a query region over a location-aware social stream. Then, we propose the *Sparse Influential Checkpoints* (SIC) framework for efficient SIM query processing. SIC maintains a sequence of *influential checkpoints* over the sliding window and each checkpoint maintains a partial solution for SIM in an append-only substream of social actions. Theoretically, SIC keeps a logarithmic number of checkpoints w.r.t. the size of the sliding window and always returns an approximate solution from one of the checkpoint for the SIM query at any time. Furthermore, we propose the *Location-based SIC* (LSIC) framework and its improved version LSIC$^+$, both of which process LSIM queries by integrating the SIC framework with a Quadtree spatial index. LSIC can provide approximate solutions for both ad hoc and continuous LSIM queries in real time, while LSIC$^+$ further improves the solution quality of LSIC. Experimental results on real-world datasets demonstrate the effectiveness and efficiency of the proposed frameworks against the state-of-the-art IM algorithms.

CCS Concepts: • **Information systems → Social networks**; *Data stream mining*; *Spatial-temporal systems*;

Additional Key Words and Phrases: Influence maximization, social network, data stream, submodular optimization, spatial index, region query

ACM Reference format:
Yanhao Wang, Yuchen Li, Ju Fan, and Kian-Lee Tan. 2018. Location-aware Influence Maximization over Dynamic Social Streams. *ACM Trans. Inf. Syst.* 36, 4, Article 43 (July 2018), 35 pages.
https://doi.org/10.1145/3230871

A preliminary version [50] of this article appeared in the Proceedings of the VLDB Endowment, Vol. 10, No. 7 (VLDB 2017).

Authors' addresses: Y. Wang, School of Computing, National University of Singapore, Singapore, 117417, Singapore; email: yanhao90@comp.nus.edu.sg; Y. Li, School of Information Systems, Singapore Management University, Singapore, 178902, Singapore; email: yuchenli@smu.edu.sg; J. Fan, School of Information, Renmin University of China, Beijing, 100872, China; email: fanj@ruc.edu.cn; K.-L. Tan, School of Computing, National University of Singapore, Singapore, 117417, Singapore; email: tankl@comp.nus.edu.sg.

1 INTRODUCTION

Social media advertising has become an indispensable tool for many companies to promote their business online. Such trends have generated \$39.94 billion in advertising revenue for Facebook in 2017 alone.[1] *Influence Maximization* (IM) is a key algorithmic problem behind viral marketing [12, 22] on social media. Through the word-of-mouth propagation among friends, IM aims to select a set of k seed users (a.k.a., *a seed set*) such that the source information (e.g., advertisement) is maximally spread in the network, and it has been extensively studied [5, 8, 9, 14, 17, 19, 21, 22, 33, 36, 37, 44–46, 49] in the past decade. Besides viral marketing, IM is also the cornerstone in many other important applications, such as sensor placement [25], rumor control [7], and social recommendation [31, 53].

Conventional IM algorithms [5, 9, 21, 22, 37, 44–46, 49] have two limitations. First, they assume the social influences among users are static and persistent. In particular, they consider the topology of the social network is fixed and the influence probabilities that are pre-trained from historical user actions in the social network [18, 24, 34, 40] remain unchanged for a long period. However, social influence is highly dynamic in reality and the propagation tendencies between users can be altered drastically by breaking news and trending topics. Consequently, it is not easy for conventional IM algorithms to capture the dynamics of social influences in real time and the seed set may be outdated. Second, they ignore the spatial information of influenced users for seed set selection. Enabled by the prevalence of positioning devices, geo-tagged user actions in social networks are massively generated from smart phones and wearable devices. The positioning information enables vast opportunities to provide new services to end users [51, 54, 56]. As such, there is an emerging demand on utilizing the geo-tagged user actions in social networks to drive the growth in location-based advertising, where the targeted users are located in a spatial region specified by an advertiser. The overall market value of location-based advertising has reached \$9.8 billion in 2015 and is expected to grow to \$29.5 billion in 2020.[2] To meet this new demand, we need to retrieve the seed set with the maximum influence in a specific region instead of the whole network, which cannot be efficiently handled by conventional IM algorithms.

There have been some research efforts on both location-aware IM [20, 26, 28, 47, 48, 57] and dynamic IM [1, 10, 38, 52, 58]. Nevertheless, existing methods do not fully satisfy the requirements of real-world propagation campaigns. Existing location-aware IM algorithms only work in the static setting where both social influences and user locations are not changed over time. Meanwhile, the state-of-the-art dynamic IM algorithms still suffer from efficiency issues to handle massive number of continuously generated social actions [38, 52]. In real-world scenarios, viral campaigns are often both time-critical and location-aware. For example, a restaurant uses online social networks to promote its one-day special offer to attract customers. It is important to target the customers who are both spatially and temporally relevant to this offer. More specifically, attracting users who are located near the restaurant and being active lately is more beneficial than attracting those who are far and silent for the promotion. To maximize market reach and attract more potential customers, it needs to identify a group of influential users with the maximal influence among targeted consumers in real time, who are used as the seed set to initiate the spread of the promotion. However, to the best of our knowledge, none of the existing techniques can handle the case where IM queries are both time-critical and location-aware.

To resolve the aforementioned drawbacks, we first propose a novel *Stream Influence Maximization* (SIM) query to track the influential users in a social network. SIM utilizes the widely available user actions (e.g., tweets/retweets) to estimate the social influences and maintains the seed set

[1]https://investor.fb.com/financials/default.aspx.
[2]http://www.biakelsey.com/location-targeted-mobile-ad-spend-reach-29-5b-u-s-2020.

continuously over streams. Specifically, SIM adopts the *sliding window* model [11], which always considers the most recent N actions. It strives to find a set of k seed users who collectively have the maximum influence value in the sliding window at any time. 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 various IM problems [19, 22, 26, 27, 47]. Furthermore, we define the *Location-aware SIM* (LSIM) query for influential users tracking over geo-tagged social streams. LSIM also adopts the sliding window model and monotone submodular influence functions. Different from SIM, it focuses on maximizing the region-constrained social influence. Given a spatial region R, LSIM only considers a subset of geo-tagged actions (e.g., tweets/retweets with locations), which are both active and located in R. LSIM returns a seed set with the largest influence value over the substream of actions in R. It is noted that the seed users may not perform any action in R themselves.

Since SIM and LSIM are NP-hard, we focus on processing them approximately with theoretical bounds. Leveraging the monotonicity and submodularity of influence functions, the greedy algorithm [35] can provide $(1 - 1/e)$-approximate solutions for both queries. However, it requires $O(k \cdot |U|)$ (where $|U|$ is the number of users in the social network) influence function evaluations for each update. Empirically, it takes around 5s to select 50 seed users from a network with 500K users, which hardly matches the rates of real-world social streams. Another related technique is *Streaming Submodular Optimization* (SSO) [3, 23]. Existing SSO algorithms [3, 23] can provide $(1/2 - \beta)$-approximate solutions for maximizing submodular functions with cardinality constraints in append-only streams. However, they cannot be directly used in the sliding window model as they do not handle the expiry of actions.

In this article, we first propose a novel *Sparse Influential Checkpoints* (SIC) framework to process SIM queries efficiently with theoretical bounds. SIC maintains a sequence of partial solutions called *influential checkpoints* with different starting points for SIM. We design a generic *Set-Stream Mapping* (SSM) interface for each checkpoint to adapt ε-approximation SSO algorithms for SIM such that a checkpoint also maintains an ε-approximate solution for SIM over its append-only sub-stream. To answer SIM for each window, SIC just returns the solution from the earliest checkpoint, which does not expire yet. Leveraging the monotonicity and subadditivity of the influence values returned by different checkpoints, given the parameter $\beta \in (0, 1)$, SIC only keeps $O(\frac{\log N}{\beta})$ influential checkpoints and maintains an $\frac{\varepsilon}{2}(1 - \beta)$-approximate solution for SIM.

Apparently, SIC does not consider the spatial information and thus cannot be used to process LSIM queries directly. Therefore, we propose the *Location-based SIC* (LSIC) framework that processes LSIM queries by integrating the SIC framework with a Quadtree spatial index. LSIC can process both continuous and ad hoc LSIM queries, but in different ways. A continuous LSIM tracks a seed set to maximize the influence value in a query region R over *a period of time*. LSIC sets up an SIC instance for each continuous LSIM and registers the instance in Quadtree. During the stream processing, if any arrival action is located in R, the instance will receive the action and update the seed set accordingly. It is guaranteed that the SIC instance processes all actions in R over the time interval and always maintains an $\frac{\varepsilon}{2}(1 - \beta)$-approximate solution for any continuous LSIM. An ad hoc LSIM requests a seed set with the maximum influence value in the query region R for the sliding window at the query time. The method for continuous LSIM cannot be applied to ad hoc LSIM, as the regions of interest are not known in advance. Thus, to process ad hoc queries, LSIC maintains SIC instances in Quadtree nodes over streams. Given an ad hoc LSIM, LSIC first finds the SIC instance that are maintained in the maximum Quadtree node covered by the query region as a partial solution. Then, it feeds the remaining actions in the query region to the partial solution to obtain the final solution. Theoretically, the final solution returned by LSIC is also $\frac{\varepsilon}{2}(1 - \beta)$-approximate for any ad hoc LSIM.

Although LSIC can efficiently process LSIM queries in real time, the seed quality of LSIC is often inferior to that of state-of-the-art static IM methods. Hence, we propose LSIC$^+$ to improve upon LSIC. LSIC$^+$ maintains the same Quadtree spatial index and SIC instances in Quadtree nodes as LSIC. But it employs an improved approximation algorithm to process LSIM queries: for any continuous or ad hoc LSIM, LSIC$^+$ first acquires the top-k' ($k' \geq k$) influential users in the query region from the Quadtree and the SIC instances in Quadtree nodes. Then, it runs a Thresholding algorithm [4] to retrieve the seed set from the top-k' users. LSIC$^+$ can achieve a better $(1 - 1/e - \gamma)$-approximation factor for any $\gamma \in (0, 1)$ if k' is sufficiently large. Empirically, the seed set returned by LSIC$^+$ is of significant higher quality than LSIC.

Finally, we evaluate the performance of the proposed frameworks on real-world datasets. First, for SIM processing, SIC can provide seed sets with at least 96% quality compared with the state-of-the-art static IM approaches while achieving up to 178× speedups over them. Second, for both continuous and ad hoc SIM processing, the seed sets of LSIC and LSIC$^+$ achieve at least 89% and 94% quality, respectively, compared with the state-of-the-art static IM approaches, and both methods demonstrate over two orders of magnitude speedups. LSIC$^+$ runs slightly slower than LSIC for continuous LSIM. Nevertheless, the average latency of LSIC$^+$ to process an ad hoc LSIM is equivalent to or even better than LSIC. Moreover, both LSIC and LSIC$^+$ process at least 5K actions per second for continuous LSIM and handle ad hoc LSIM queries within 150ms, which can meet the demand for real-time query processing in real-world social streams.

Our main contributions in this article are summarized as follows.

- We address the limitations of existing IM methods for fast evolving and location-based social networks and define the SIM and LSIM queries over sliding windows.
- We develop the SIC framework for efficient SIM processing. Theoretically, SIC provides $\frac{\varepsilon}{2}(1 - \beta)$-approximate solutions for SIM.
- We devise LSIC and LSIC$^+$ to process continuous and ad hoc LSIM queries efficiently. LSIC can provide solutions with the same $\frac{\varepsilon}{2}(1 - \beta)$-approximation for LSIM. LSIC$^+$ substantially improves the seed quality of LSIC and achieves a $(1 - 1/e - \gamma)$-approximation factor for LSIM when k' is sufficiently large.
- We demonstrate the effectiveness and efficiency of the proposed frameworks by extensive experiments on real-world datasets.

The rest of this article is organized as follows. Section 2 reviews the related work. Section 3 formally defines the SIM and LSIM queries over sliding windows. Then, Section 4 presents the SIC framework for SIM processing. Next, Section 5 introduces the LSIC and LSIC$^+$ frameworks for LSIM processing. Section 6 reports the experimental setup and results. Finally, Section 7 concludes the whole article and discusses the future work.

2 RELATED WORK

We summarize the literature related to this work from three areas, namely influence maximization, streaming submodular optimization, and function estimation over sliding windows.

Influence Maximization in Social Networks. There has been a vast amount of literature on influence maximization (IM) [5, 8–10, 16, 17, 19–22, 26, 33, 36–38, 44–47, 49, 52, 58] over the last decade (see Reference [30] for an extensive survey). Kempe et al. [22] are among the first to formulate IM as an optimization problem. They prove that the influence functions are monotone and submodular under two classical diffusion models, i.e., Independent Cascade (ICM) and Linear Threshold (LTM). Due to the NP-hardness of IM, they propose a $(1 - 1/e - \varepsilon)$-approximation greedy framework for IM. After the seminal work, many techniques have been proposed to improve the efficiency of the greedy framework. The state-of-the-art static IM method under ICM and LTM is *Reverse Influence Sampling* (RIS) [5, 21, 37, 44, 45, 49]. IMM [44] is a typical RIS-based algorithm for IM. It improves the efficiency of the original RIS [5] algorithm by reducing the sampling complexity and achieves nearly linear running time w.r.t. the graph size while retaining the $(1-1/e-\varepsilon)$-approximation ratio. In the experiments, we use IMM as the baseline for static IM.

Recently, IM is extended to various location-aware problems by considering different spatial contexts [20, 26, 28, 41, 47, 48, 57]. Li et al. [26] first define a location-aware IM problem by finding a seed set to maximize the influence spread over users in a query region. They adopt the PMIA [9] model for influence estimation and propose efficient algorithms to provide $\varepsilon(1-1/e)$-approximate solutions for location-aware IM under PMIA. Li et al. [28] model users' geographical preferences based on their check-in behaviors and propose a location-aware IM problem to maximize the influence weighted by users' preferences to the query region. They propose a community-based heuristic algorithm for the problem. Wang et al. [47, 48] propose distance-aware IM to maximize the influence w.r.t. a query position. They consider the influence over users decrease with their distances to the query position and aim to find the seeds to maximize the distance-aware social influence. They extend both PMIA [47] and RIS [48] based methods for the problem. Zhou et al. [57] study the location-based IM problem in the Online to Offline (O2O) environment. Guo et al. [20] study the IM problem in case of trajectory databases.

However, the above methods for both conventional and location-aware IM are based on static social influences. They cannot efficiently support highly evolving networks, because a complete rerun is required for every update. Meanwhile, there are emerging studies about dynamic IM on evolving graphs. Chen et al. [10] propose an Upper Bound Interchange (UBI) heuristic to track the seed set over a sequence of snapshot graphs. Nevertheless, UBI cannot provide any theoretical guarantee for seed quality and its performance degrades dramatically when the size of the seed set increases [50]. Dynamic IM algorithms with theoretical bounds are presented in References [38, 52]. They maintain RIS-based [5] indices against graph changes and guarantee the same $(1-1/e-\varepsilon)$-approximation for IM. Due to the high maintenance cost, they only process several hundred graph updates per second, which cannot meet the requirement of real-world social streams. Subbian et al. [43] propose a method to compute users' influence scores and identify the influencers over social streams. Different from IM, they define a top-k query to track a set of users with the highest influence scores over social streams without considering the influence overlaps between different users. There are several studies on incorporating temporal dynamics into the IM problem. For example, Gomez-Rodriguez et al. [16, 17] and Tang et al. [44] study IM under the continuous-time diffusion model. However, both studies focus on modeling the temporal dynamics of the diffusion process and maximizing the influence spread within a given time constraint. They do not take into account the evolving of social influences over time, and thus are orthogonal to SIM and LSIM proposed in this article.

Streaming Submodular Optimization(SSO). Existing SSO [2, 3, 23, 39] algorithms adopt the append-only streaming model where elements arrive one by one and dynamically maintain a set of at most k elements to maximize a submodular function w.r.t. the observed elements. Saha et al. [39] and Ausiello et al. [2] develop two methods for a special case of SSO (i.e., the online Maximum k-Coverage problem) with the same 1/4-approximation. The state-of-the-art algorithms for SSO are SieveStreaming [3] and ThresholdStream [23], both of which achieve a $(1/2-\beta)$-approximation for any $\beta > 0$. Unfortunately, these algorithms cannot be directly applied to the *sliding window* model, because they do not handle the continuous expiry of elements. Nevertheless, we will show that they can serve as checkpoint oracles for SIC in Section 4.1.

Very recently, Epasto et al. [13] propose algorithms for SSO over sliding windows. Although the $(1/3-\beta)$-approximation algorithm in Reference [13] is similar to SIC, they have two major differences. First, the algorithm is designed for the Set-Stream model instead of social action streams. Second, it is specific for ThresholdStream and cannot be adapted to other SSO algorithms. Different from Reference [13], SIC adopts the *Set-Stream Mapping* (SSM) interface to handle social action streams and can use different SSO algorithms as checkpoint oracles.

Function Estimation over Sliding Windows. Several techniques [6, 11] are proposed to continuously estimate a function over sliding windows. They leverage some special properties of target functions to achieve sublinear performance and provable quality. Let g be the target function and A, B, C be three sequences in a stream such that B is a tail subsequence of A and C is contiguous to B. The exponential histogram [11] is proposed to approximate *weakly additive* functions, i.e., $g(A) + g(C) \leq g(A \cup C) \leq C_f(g(A) + g(C))$ for some $C_f \geq 1$. The smooth histogram [6] requires the target functions to be (α, β)-*smooth*. Specifically, g is (α, β)-*smooth* if $\frac{g(B)}{g(A)} \geq 1 - \beta$ implies $\frac{g(B \cup C)}{g(A \cup C)} \geq 1 - \alpha$ for some $0 < \beta \leq \alpha < 1$. Following the analysis in Reference [13], 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 article, we adopt monotone submodular functions [22] that are widely used in the social influence analysis. However, such functions are not *weakly additive* and no SSO algorithm can achieve an approximation ratio of better than $(1 - 1/e) \approx 0.63$ unless P=NP. This implies that both techniques cannot be directly applied to our scenario.

3 PROBLEM DEFINITION

3.1 Influence in Social Streams

In this subsection, we introduce the concept of *influence* in social action streams, formally define the *influence set* of a user or a set of users in the *sliding window* model, and consider the *influence function* defined on the *influence set*.

Let A be a social stream over a social network with a set U of users. The social stream A comprises unbounded time-sequenced social actions $\{a_1, a_2, \ldots\}$, 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: *A user u performs a_t at time t responding to an earlier action $a_{t'}$*. If an action a_t does not respond to any previous action, e.g., a user u posts original content, then we call it a *root action* and denote it by $a_t = \langle u, nil \rangle_t$. In a location-based social network, a_t is associated with a spatial location $\mathbf{p}_t(x_t, y_t) \in \mathbb{R}^2$ representing that a_t is performed at $\mathbf{p}_t$. For example, "tweet" and "retweet" on Twitter are typical social actions. A tweet/retweet will have a geo-tag if the user posts it from a geo-position enabled device and allows for position sharing.

Like many data streams, social streams are time-sensitive: recent actions are more valuable than older ones. We adopt the well-recognized *sequence-based*[3] *sliding window* [11] 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 ith $(i \in [1, N])$ action within W_t. Then, we use $U_t \subseteq U$ to denote the set of active users who perform at least one action in W_t, i.e., $U_t = \{W_t[i].u | i = 1, \ldots, N\}$.

Since social actions can directly reflect the influence diffusion in the social network [18, 19, 24, 43], we define the *influence* between users according to their performed actions. We say u *influences* v in W_t, denoted by $(u \rightsquigarrow v)_t$, if there exists an action a performed by user v such that $a \in W_t$ and a is *directly* or *indirectly* triggered by an action a' of u. Note that such an a' is not necessarily in W_t.

We formally define the *influence set* of a user as follows.

[3]We only consider the sequence-based sliding window model in this article. But it is noted that the proposed frameworks can be adapted to the time-based sliding window model trivially.

Definition 3.1 (Influence Set). The influence set of a user $u \in U$ at time t, denoted as $I_t(u) \subseteq U_t$, is the set of users influenced by u w.r.t. 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 perform actions under the impact of u. The concept of the influence set can be naturally extended to a set of users. In particular, let $S = \{u_1, \dots, u_k\}$ be a set of k users, the *influence set* of S w.r.t. W_t is a union of the influence sets of all its members, i.e., $I_t(S) = \cup_{u \in S} I_t(u)$.

Furthermore, we define the *region-constrained influence set* in the context of geo-tagged social action streams.

Definition 3.2 (Region-constrained Influence Set). Given a region $R = [\mathbf{p}^l(x^l, y^l), \mathbf{p}^h(x^h, y^h)] \subseteq \mathbb{R}^2$, the *influence set* $I_t(u, R)$ of u at time t in R is a subset of $I_t(u)$ consisting of the users who perform an action a_t such that $\mathbf{p_t} \in R$ (i.e., $x_t \in [x^l, x^h] \wedge y_t \in [y^l, y^h]$) and a_t is triggered by $a_{t'}$ of u. The concept of region-constrained influence sets for a set of users can be defined similarly, i.e., $I_t(S, R) = \cup_{u \in S} I_t(u, R)$.

Given a set S of users, the influence function is defined on its influence set $I_t(S)$. The influence value of S is measured by a set function $f(I_t(\cdot)) : 2^{|U|} \to \mathbb{R}_{\geq 0}$, which is nonnegative, monotone[4] and submodular[5]. Such functions are widely accepted as the influence functions for IM due to its natural representation of the "diminishing returns" property of the social influence [22]. We also extend the definition of the influence function to region-constrained influence sets. Given a set S of users and a region R, the influence value of S in R is given by a function $f(I_t(\cdot, \cdot)) : 2^{|U|} \times (\mathbb{R}^2, \mathbb{R}^2) \to \mathbb{R}_{\geq 0}$. Obviously, given any specific region R, the location-aware influence function $f(I_t(\cdot, R))$ retains the monotonicity and submodularity of $f(I_t(\cdot))$.

For ease of presentation, we consider the cardinality functions, i.e., $f(I_t(S)) = |I_t(S)|$ and $f(I_t(S, R)) = |I_t(S, R)|$, as the influence functions in the remaining of this article. It is noted that any other monotone submodular influence functions can also be adopted similarly. Example 3.3 illustrates the concept and computation of *social influences* over sliding windows.

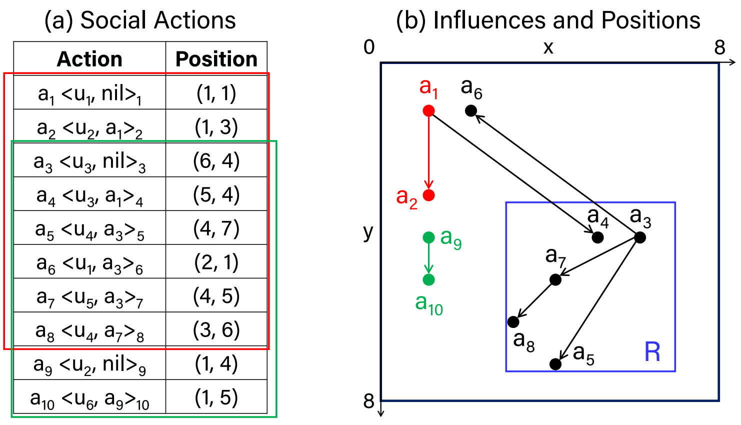
Example 3.3. We consider an example for social stream as shown in Figure 1. Given the window size $N = 8$, the sliding windows at time 8 and 10, i.e., W_8 and W_{10}, are highlighted by red and green boxes, respectively. The influence set of u_1 at time 8 is $I_8(u_1) = \{u_1, u_2, u_3\}$ as a_1, a_6 are performed by u_1 and a_2, a_4 performed, respectively, by u_2, u_3 are triggered by a_1 in W_8. When the window slides from W_8 to W_{10}, a_1 and a_2 expire while a_9 and a_{10} arrive. Then, $I_{10}(u_1) = \{u_1, u_3\}$, because u_2 is deleted from $I_{10}(u_1)$ for the expiry of a_2. However, since a_4 has not expired yet, u_1 still *influences* u_3 in W_{10} regardless of the expiry of a_1.

The social actions in Figure 1 are associated with positions in two-dimensional space $[0, 8]^2$. Given a region $R = [(3, 3), (7, 7)]$ (see the blue box in Figure 1(b)), the influence set of u_1 is $I_8(u_1, R) = \{u_3\}$, because $\mathbf{p}_4 \in R$ and $\mathbf{p}_1, \mathbf{p}_6 \notin R$. When the window slides from W_8 to W_{10}, $I_{10}(u_1, R) = \{u_3\}$ remains unchanged as a_4 does not expire at time 10 and $\mathbf{p}_9, \mathbf{p}_{10} \notin R$.

3.2 Influence Maximization Query

In this subsection, we define the *Stream Influence Maximization* (SIM) and *Location-aware Stream Influence Maximization* (LSIM) queries over social streams based on the concepts introduced in Section 3.1. We then prove the NP-hardness of retrieving the optimal solutions for both queries.

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


(a) Social Actions

Action	Position
a_1 <u_1, nil>$_1$	(1, 1)
a_2 <u_2, a_1>$_2$	(1, 3)
a_3 <u_3, nil>$_3$	(6, 4)
a_4 <u_3, a_1>$_4$	(5, 4)
a_5 <u_4, a_3>$_5$	(4, 7)
a_6 <u_1, a_3>$_6$	(2, 1)
a_7 <u_5, a_3>$_7$	(4, 5)
a_8 <u_4, a_7>$_8$	(3, 6)
a_9 <u_2, nil>$_9$	(1, 4)
a_{10} <u_6, a_9>$_{10}$	(1, 5)

Fig. 1. Example for geo-tagged social action stream. In Figure (a), we list a sequence of 10 geo-tagged social actions. In Figure (b), we illustrate the spatial information and social influence of these actions. The actions are plotted at the coordinates of their positions and the influence relationships are represented by arrows.

As new actions arrive at high speed while old ones expire at the same rate, users with the largest influence values keep evolving over time. To track the influential users over social streams in real time, we propose the Stream Influence Maximization (SIM) query as follows.

Definition 3.4 (SIM). Let W_t be the sliding window at time t. Given $k \in \mathbb{Z}^+$, an SIM $q_t(k)$ is a query that returns a set of at most k seed users S_t^* (we use $OPT_t = f(I_t(S_t^*))$ to denote its influence value) who collectively have the maximum influence value w.r.t. W_t, i.e., $S_t^* = \text{argmax}_{S \subseteq U:|S| \leq k} f(I_t(S))$.

Extending SIM in the context of geo-tagged social streams, we define the Location-aware Stream Influence Maximization (LSIM) query as follows.

Definition 3.5 (LSIM). Let W_t be the sliding window at time t. Given $k \in \mathbb{Z}^+$ and a region R, an LSIM $q_t(k, R)$ is a query that returns a set of at most k seed users[6] $S_{t,R}^*$ (we use $OPT_{t,R} = f(I_t(S_{t,R}^*, R))$ to denote its influence value) who collectively have the maximum influence value in R w.r.t. W_t, i.e., $S_{t,R}^* = \text{argmax}_{S \subseteq U:|S| \leq k} f(I_t(S, R))$.

We consider both ad hoc and continuous LSIM queries in this article. An ad hoc query $q_t(k, R)$ at time t retrieves the set of seed users who have the largest influence in R w.r.t. W_t. A continuous query $q_{[t_1,t_2]}(k, R)$ registered at t_1 and unregistered at t_2 tracks the set of seed users who have the maximum influence in R at any time t from t_1 to t_2.

We continue with the example in Figure 1 to show how SIM and LSIM keep track of the most influential users over the sliding window.

Example 3.6. An SIM query $q_8(2)$ returns $S_8^* = \{u_1, u_3\}$, because $I_8(S_8^*) = I_8(u_1) \cup I_8(u_3)$ contains all users in U_8. We have $OPT_8 = f(I_8(S_8^*)) = 5$ if the cardinality function is used. After a_1, a_2 expire and a_9, a_{10} arrive at time 10, $f(I_{10}(S_8^*)) = 4$ as u_2 is deleted from $I_{10}(S_8^*)$. Thus, $q_{10}(2)$ returns $S_{10}^* = \{u_2, u_3\}$ for W_{10}, because $I_{10}(S_{10}^*)$ contains all users in U_{10}. We have $OPT_{10} = 6$ accordingly.

Given $R = [(3,3), (7,7)]$, an LSIM query $q_8(2, R)$ returns $S_{8,R}^* = \{u_3\}$, because u_3 influences all users who perform actions in R w.r.t. W_8. We get $OPT_{8,R} = 3$. In addition, $S_{10,R}^* = \{u_3\}$ and $OPT_{10,R} = 3$ are not changed from time 8 to 10.

[6]It is noted that the seed users may not perform any action themselves in region R.

Table 1. Frequently Used Notations

Symbol	**Definition and Description**
U	the set of users in a social network
$A = \{a_1, a_2, \ldots\}$	an unbounded sequence of social actions in a social network
$a_t = \langle u, a_{t'} \rangle_t$	an action $a_t \in A$ performed by $u \in U$ at time t triggered by $a_{t'}$ $(t' < t)$
$\mathbf{p}_t(x_t, y_t)$	a_t is performed at $\mathbf{p}_t$ with longitude x_t and latitude y_t
N	the size of the sliding window
$W_t, W_t[i]$	the sliding window at time t and the ith action in W_t
R	A region $R = [\mathbf{p}^l, \mathbf{p}^h]$ denoting a rectangle defined by diagonal corners $\mathbf{p}^l(x^l, y^l)$ and $\mathbf{p}^h(x^h, y^h)$
$I_t(u), I_t(S)$	the influence set of a user u or a set of users S w.r.t. W_t
$I_t(u, R), I_t(S, R)$	the region-constrained influence set of u or S in R w.r.t. W_t
$I_t[i](\cdot)$	the influence set of u or S for contiguous actions $\{W_t[i], \ldots, W_t[N]\}$
$f(I_t(\cdot)), f(I_t(\cdot, \cdot))$	the location-unaware and location-aware influence functions
k	the maximum size of the seed set for SIM and LSIM
$q_t(k)$	An SIM query at time t with a cardinality constraint k
$q_t(k, R)$	An LSIM query at time t with a cardinality constraint k and a query region R
$S_t^*, S_{t,R}^*$	the optimal seed sets of $q_t(k)$ and $q_t(k, R)$. We use OPT_t and $OPT_{t,R}$ to denote their influence values, respectively.
$\Lambda_t[i]$	an influential checkpoint maintaining an ε-approximate solution for SIM w.r.t. contiguous actions $\{W_t[i], \ldots, W_t[N]\}$
$S_t^*[i]$	the optimal seed set of SIM w.r.t. contiguous actions $\{W_t[i], \ldots, W_t[N]\}$. We use $OPT_t[i]$ to denote its influence value.

Then, we show the NP-hardness of SIM by reducing a well-known NP-hard problem, i.e., the *Maximum k-Coverage* problem [15], to SIM in polynomial time. The NP-hardness of LSIM can be proved accordingly, because SIM is a special case of LSIM when R is the full space.

THEOREM 3.7. *SIM and LSIM are NP-hard.*

PROOF. We prove Theorem 3.7 by reducing a well-known NP-hard problem, i.e., *Maximum k-Coverage*, to SIM. A *Maximum k-Coverage* instance consists of an positive integer k and a collection of m non-empty sets $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$. It aims to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq k$ and $| \cup_{s_i \in \mathcal{S}'} s_i|$ is maximized. Any *Maximum k-Coverage* instance is reduced to a *SIM* instance as follows: Let t be the timestamp and $G = \cup_{s_i \in \mathcal{S}} s_i$ be the ground set. For any set $s_i \in \mathcal{S}$, we create an action $a_{t'} = \langle u_i, nil \rangle_{t'}$ where $u_i \notin G$. For each element $e \in s_i$, we create an action $a_t = \langle e, a_{t'} \rangle_t$ $(t' < t)$. Next, we stream all created actions to SIM in the ascending order of t. Since the total number of created actions is $O(m \cdot |G|)$, the reduction is performed in polynomial time. Let the window size N equal to the total number of created actions. Let $S_N^* = \{u_i, \ldots, u_j\}$ be the optimal solution of SIM w.r.t. W_N using the cardinality function. $\{s_i, \ldots, s_j\}$ will be optimal for the corresponding *Maximum k-Coverage* instance as well. Thus, the reduction naturally follows. Finally, because SIM is a special case of LSIM when R is the full space, LSIM is NP-hard as well. □

Before moving on to the remaining parts, we summarize the frequently used notations in Table 1.

4 SPARSE INFLUENTIAL CHECKPOINTS

In this section, we present the *Sparse Influential Checkpoints* (SIC) framework for SIM. We first describe the Set-Stream Mapping (SSM) interface for SIM over an append-only action stream in Section 4.1. Then, we introduce the idea of the SIC framework, demonstrate its maintenance procedure, and analyze it theoretically in Section 4.2.

4.1 Set-Stream Mapping

A simple approach to SIM is the Greedy algorithm [35] for submodular maximization with cardinality constraints. Because the influence function is monotone and submodular, Greedy can return a $(1-1/e)$-approximate solution for SIM. As Greedy does not store any intermediate result, it is required to be rerun from scratch for each window slide. When the window slides from W_{t-1} to W_t, we first update the influence sets of users according to the arrival and expired actions. Then, Greedy starts with an empty user set $S_0 = \emptyset$. At each iteration i $(1 \leq i \leq k)$, it adds a user u to S_{i-1} maximizing $f(I_t(S_{i-1} \cup \{u\})) - f(I_t(S_{i-1}))$. After k iterations, S_k is returned the result. Although it achieves the best approximation ratio, i.e., $(1-1/e)$, for submodular maximization with cardinality constraints [35], $O(k \cdot |U|)$ influence function evaluations are needed for every update. Such an inefficient scheme cannot handle a large window size with new actions arriving rapidly.

To improve the efficiency, we first propose a method to process SIM incrementally with only one pass over social action streams. We propose to use SSO algorithms [2, 3, 23, 39] in the *append-only* streaming model for SIM. However, they are designed for the set-stream model where elements in the stream are sets instead of actions. Generally, an algorithm $\mathcal{A}$ over 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., $X_1, \ldots, X_t$). Given a stream of sets $\{X_1, X_2, \ldots, X_m\}$, the objective of $\mathcal{A}$ is to maximize $f'(CX_t)$ at any time t $(1 \leq t \leq m)$. Although resembling our problem, the set-stream 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 arrival actions may trigger updates in the influence sets of existing users.

To bridge the gap between the two stream models and leverage existing SSO algorithms based on the set-stream model, we propose a generic Set-Stream Mapping (SSM) interface. Let an influential checkpoint $\Lambda_t[i]$ $(1 \leq i \leq N)$ denote a *checkpoint oracle* that provides an ε-approximate solution for SIM over contiguous actions $\{W_t[i], \ldots, W_t[N]\}$. Next, we will show how SSM adapts an SSO algorithm $\mathcal{A}$ for the set-stream model to serve as a checkpoint oracle $\Lambda_t[i]$. 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, each $\Lambda_t[i]$ takes the following steps:

(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 solution quality, update complexity, and function generality. Typical oracles are listed in Table 2. An important conclusion is that the SSM procedure does not affect the quality guarantee of the mapped algorithms.

Theorem 4.1. *Let $\mathcal{A}$ be an ε-approximation SSO algorithm in the set-stream model and $\overline{\mathcal{A}}$ be the mapped algorithm of $\mathcal{A}$ using SSM. $\overline{\mathcal{A}}$ is ε-approximation for SIM.*

Table 2. Candidate Checkpoint Oracles for IC and SIC

Oracle	**Approximation Factor**	**Update Complexity**	**Function**
SieveStreaming [3]	$1/2 - \beta$	$O(\log k/\beta)$	General
ThresholdStream [23]	$1/2 - \beta$	$O(\log k/\beta)$	General
Blog Watch [39]	$1/4$	$O(k)$	Cardinality
MkC [2]	$1/4$	$O(k \log k)$	Cardinality

Note that we use SieveStreaming as the exemplar oracle in the remainder of this article.

Proof. To show $\overline{\mathcal{A}}$ is ε-approximation for SIM, 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 w.r.t. 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 set-stream as independent sets regardless of whether they belong to the same user. We run $\overline{\mathcal{A}}$ over 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 can only 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., $f(\cup_{X \in CX_t} X) \geq \varepsilon OPT_t^*$. As the up-to-date influence sets always appear in the append-only set-stream, we have $OPT_t^* \geq OPT_t$, and thus $f(I_t(U_t)) \geq \varepsilon OPT_t$. Therefore, U_t is an ε-approximate solution for SIM w.r.t. W_t. □

Given the result of Theorem 4.1, an influential checkpoint $\Lambda_t[i]$ is guaranteed to maintain an ε-approximate solution for SIM over contiguous actions $\{W_t[i], \ldots, W_t[N]\}$. We can devise a simple framework (i.e., Influence Checkpoints (IC) in [50]) for SIM over sliding windows. IC maintains N influential checkpoints for every action in W_t at any time t. For each window slide, it simply deletes the checkpoint corresponding to the expired action and creates a new checkpoint for the arrival action a_t. Then, each existing checkpoint processes a_t accordingly. At any time t, the first non-expired checkpoint, i.e., $\Lambda_t[1]$, must have processed all actions in W_t and maintains an ε-approximate solution for SIM w.r.t. W_t.

According to the SSM steps, an action a_t is mapped to at most d influence sets, where d is the number of predecessors of a_t in the propagation. In practice, d is usually small, e.g., $d < 5$ on average as shown in our experiments (see Table 3). Since the number of checkpoints in IC is N, the total number of checkpoint evaluations is $O(dN)$. If the update complexity of the checkpoint oracle for each set is $O(g)$, then the total time complexity of IC for each update is $O(dgN)$. Using SieveStreaming as an example oracle, we can see IC is $(1/2 - \beta)$-approximate and has $O(\frac{dN \log k}{\beta})$ update complexity. A detailed description of the SieveStreaming algorithm and the IC framework is omitted in this article and can be found in References [3] and [50], respectively.

4.2 The Sparse Influential Checkpoints Framework

As real-world applications often require millions of actions in one window, IC still incurs prohibitive cost to maintain N checkpoints in practice. To reduce the number of checkpoints 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 when the window slides. Specifically, the number of checkpoints maintained by SIC is logarithmic to N while its approximation ratio remains $\frac{\varepsilon}{2}(1 - \beta)$ for any $\beta > 0$ if the checkpoint oracle is ε-approximate.

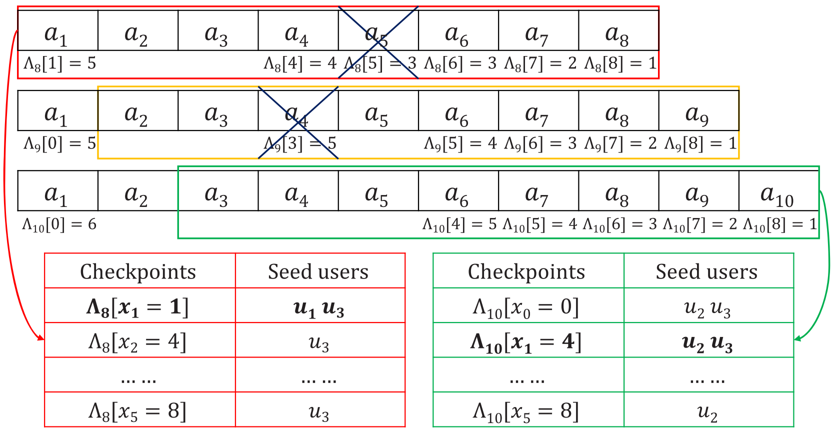
ALGORITHM 1: Sparse Influential Checkpoints

```
Input: SIC at time $t-1$: $\{\Lambda_{t-1}[x_0], \Lambda_{t-1}[x_1], \dots, \Lambda_{t-1}[x_s]\}$, a parameter $\beta > 0$
Output: The solution for $q_t(k)$ at any time $t$
1  while receiving a new action $a_t$ at time $t$ do
2      Create $\Lambda_t[x_{s+1}]$ where $x_{s+1} = N$, $s \leftarrow s + 1$;
3      foreach $\Lambda_{t-1}[x_i]$ do $\Lambda_t[x_i] \leftarrow \Lambda_{t-1}[x_i]$, $x_i \leftarrow x_i - 1$;
4      foreach $\Lambda_t[x_i]$ do $\Lambda_t[x_i]$.process($a_t$);
5      for $i \in [1, s]$ do
6          Initialize $\Lambda^- \leftarrow \emptyset$;
7          for $j \in [i+1, s-1]$ do
8              if $\Lambda_t[x_j] \geq (1-\beta)\Lambda_t[x_i]$ and $\Lambda_t[x_{j+1}] \geq (1-\beta)\Lambda_t[x_i]$ then
9                  $\Lambda^- \leftarrow \Lambda^- \cup \{\Lambda_t[x_j]\}$;
10             else
11                 break;
12         Delete the checkpoints in $\Lambda^-$ from SIC andshift the remaining checkpoints accordingly;
13     if $x_1 = 0$ then
14         Delete $\Lambda_t[x_0]$ and shift the remaining checkpoints accordingly;
       /* On processing $q_t(k)$ at time $t$                                    */
15     Retrieve the solution of $\Lambda_t[x_1]$ for $q_t(k)$;
```

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 $\{\Lambda_t[x_0], \Lambda_t[x_1], \dots, \Lambda_t[x_s]\}$ 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 $OPT_t[x_i]$ 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 approximation ratio should be at least $\frac{\varepsilon}{2}(1-\beta)$ at any time $t' > t$. We will analyze the theoretical soundness of SIC after introducing its maintenance procedure.

Algorithm 1 presents how to efficiently maintain the checkpoints over sliding windows in SIC. 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 and 4). Then the efficient deletion of checkpoints are presented in Lines 5–12. 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 13 and 14). It is noted 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 1 always maintains a bounded ratio between two neighboring checkpoints, a bounded approximation ratio is guaranteed by using $\Lambda_t[x_1]$ as the result for the current window. We provide a running example for the checkpoint maintenance of SIC in Example 4.2.


a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
$\Lambda_8[1] = 5$ $\Lambda_8[4] = 4$ $\Lambda_8[5] = 3$ $\Lambda_8[6] = 3$ $\Lambda_8[7] = 2$ $\Lambda_8[8] = 1$
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
$\Lambda_9[0] = 5$ $\Lambda_9[3] = 5$ $\Lambda_9[5] = 4$ $\Lambda_9[6] = 3$ $\Lambda_9[7] = 2$ $\Lambda_9[8] = 1$
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10}
$\Lambda_{10}[0] = 6$ $\Lambda_{10}[4] = 5$ $\Lambda_{10}[5] = 4$ $\Lambda_{10}[6] = 3$ $\Lambda_{10}[7] = 2$ $\Lambda_{10}[8] = 1$

Checkpoints	Seed users
$\boldsymbol{\Lambda_8[x_1 = 1]}$	$\boldsymbol{u_1\, u_3}$
$\Lambda_8[x_2 = 4]$	u_3
... ...	... ...
$\Lambda_8[x_5 = 8]$	u_3

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

Fig. 2. Example for checkpoint maintenance in SIC. We continue with the example in Figure 1 and illustrate how SIC maintains the checkpoints at times 8, 9, and 10 (the deleted ones are marked by crosses).

Example 4.2. An example for SIC maintenance is illustrated in Figure 2. Let $N = 8$, $k = 2$ and $\beta = 0.3$. There are initially 6 checkpoints in SIC at time 8. According to Algorithm 1, $\Lambda_8[5]$ is deleted from SIC, since $\Lambda_8[6] = 3 > (1 - 0.3) \times 4 = (1 - \beta)\Lambda_8[4]$. At time 8, the seed users for $q_8(2)$ are $\{u_1, u_3\}$ returned by $\Lambda_8[1]$. When the window slides 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 not deleted, because the second checkpoint $\Lambda_9[3]$ has not expired yet. Then, all checkpoints will be updated according to a_9. After the update procedure, $\Lambda_9[3]$ can be deleted, because $\Lambda_9[5] > (1 - \beta)\Lambda_9[0]$. Finally, all checkpoints are updated according to a_{10} at time 10 and no checkpoint is deleted. The first non-expired checkpoint $\Lambda_{10}[4]$ is used for $q_{10}(2)$ at time 10 and $\{u_2, u_3\}$ is returned as the seed set.

Theoretical Analysis of SIC. In the following, we will demonstrate the theoretical soundness of SIC and analyze the complexity of SIC. 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 4.3 (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 $\{a_{t_a}, \dots, a_{t_b}\}$ with the corresponding checkpoint denoted as $\Lambda_{t_b}^{t_a}$. Given any three timestamps $t_1 \leq t_2 \leq t_3$, a checkpoint oracle is monotone if $\Lambda_{t_3}^{t_1} \geq \Lambda_{t_2}^{t_1}$. In addition, a checkpoint oracle is subadditive if $\Lambda_{t_3}^{t_1} \leq \Lambda_{t_2}^{t_1} + \Lambda_{t_3}^{t_2}$.

Lemma 4.4. *Let $t_a \leq t_b$ be two timestamps and $OPT_{t_b}^{t_a}$ denote the optimal oracle (also the optimal influence value) for $W_{t_b}^{t_a}$. The optimal oracle is monotone and subadditive.*

Proof. Let $I_{t_b}^{t_a}(S)$ be the influence set of S and S_{t_a,t_b}^* be the optimal solution of SIM for contiguous actions in $W_{t_b}^{t_a}$. Because S_{t_1,t_2}^* must be a candidate solution for $W_{t_3}^{t_1}$, it is obvious that $OPT_{t_3}^{t_1} \geq OPT_{t_2}^{t_1}$. For any $S \subseteq U$, we have $I_{t_3}^{t_1}(S) = I_{t_2}^{t_1}(S) \cup I_{t_3}^{t_2}(S)$. Then, we have $OPT_{t_3}^{t_1} = f(S_{t_1,t_3}^*) \leq f(I_{t_2}^{t_1}(S_{t_1,t_3}^*)) + f(I_{t_3}^{t_2}(S_{t_1,t_3}^*)) \leq OPT_{t_2}^{t_1} + OPT_{t_3}^{t_2}$, where the first inequality holds for the property of influence sets, and the second inequality is satisfied, because S_{t_1,t_3}^* is a candidate solution for both $W_{t_2}^{t_1}$ and $W_{t_3}^{t_2}$. Finally, we prove that $OPT_{t_3}^{t_1} \leq OPT_{t_2}^{t_1} + OPT_{t_3}^{t_2}$. □

We note that, although the optimal checkpoint oracle is both monotone and subadditive, it is intractable unless P=NP. In practice, we use 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 result of Lemma 4.4 and the monotonicity of approximate checkpoint oracles, we show that the checkpoints maintained by Algorithm 1 is theoretically bounded for SIM.

Lemma 4.5. *Given any four timestamps $t_1 \leq t_2 \leq t_3 \leq t_4$, if $(1-\beta)\Lambda_{t_3}^{t_1} \leq \Lambda_{t_3}^{t_2}$, we have $\frac{\varepsilon}{2}(1-\beta)OPT_{t_4}^{t_1} \leq \Lambda_{t_4}^{t_2}$ for any $\beta \in (0,1)$.*

Proof. We can acquire the following inequalities $\Lambda_{t_4}^{t_2} \geq \frac{1}{2}(\Lambda_{t_3}^{t_2} + \Lambda_{t_4}^{t_2}) \geq \frac{1}{2}((1-\beta)\Lambda_{t_3}^{t_1} + \Lambda_{t_4}^{t_2}) \geq \frac{\varepsilon}{2}(1-\beta)(OPT_{t_3}^{t_1} + OPT_{t_4}^{t_2}) \geq \frac{\varepsilon}{2}(1-\beta)OPT_{t_4}^{t_1}$, because the first inequality holds from the monotonicity of the approximate checkpoint oracles, the second inequality is due to $\Lambda_{t_3}^{t_2} \geq (1-\beta)\Lambda_{t_3}^{t_1}$, the third inequality holds, because the checkpoint oracle is ε-approximate, and the final inequality holds, since the optimal checkpoint oracle is monotone and subadditive. □

According to Lemma 4.5, if $(1-\beta)\Lambda_{t_3}^{t_1} \leq \Lambda_{t_3}^{t_2}$, using the checkpoint oracle started at time t_2 to approximate any checkpoints between t_1 and t_2 can always achieve an $\frac{\varepsilon}{2}(1-\beta)$-approximation ratio for any number of appending actions. Next, we present Lemma 4.6 to demonstrate the property of the checkpoints maintained by Algorithm 1.

Lemma 4.6. *SIC for W_t contains a sequence of s checkpoints $\Lambda_t[x_0], \Lambda_t[x_1], \ldots, \Lambda_t[x_s]$ where $x_0 < x_1 < \cdots < x_s$ maintained by Algorithm 1. Given $\beta \in (0,1)$, any neighboring checkpoints $\Lambda_t[x_i]$, $\Lambda_t[x_{i+1}]$ and $\Lambda_t[x_{i+2}]$ satisfy one of the following three conditions:*

(1) if $\Lambda_t[x_{i+1}] \geq (1-\beta)\Lambda_t[x_i]$, then $\Lambda_t[x_{i+2}] < (1-\beta)\Lambda_t[x_i]$ or $x_{i+1} = t$.
(2) if $x_{i+1} \neq x_i + 1$ and $\Lambda_t[x_{i+1}] < (1-\beta)\Lambda_t[x_i]$, then $\frac{\varepsilon}{2}(1-\beta)OPT_t[x_i] \leq \Lambda_t[x_{i+1}]$.
(3) $x_{i+1} = x_i + 1$ and $\Lambda_t[x_{i+1}] < (1-\beta)\Lambda_t[x_i]$.

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.

Then, assuming Lemma 4.6 holds at time t, and we show that it still holds after the update procedure in Algorithm 1 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 $t+1$. $\Lambda_t[x_{i+1}]$ is the subsequent checkpoint of $\Lambda_{t+1}[x_i]$ at time t. Next, we discuss all possible cases for SIC maintenance at time $t+1$.

Case 1. $x_{i+1} \neq x_i + 1$ and $\Lambda_t[x_{i+1}]$ is deleted at time $t+1$. 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 5–12 of Algorithm 1. Thus, Condition 1 holds at time $t+1$.

Case 2. $x_{i+1} \neq x_i + 1$ and $\Lambda_{t+1}[x_{i+1}]$ is not deleted at time $t+1$. 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 time t', we have $\Lambda_{t'}[x_{i+1}] \geq (1-\beta)\Lambda_{t'}[x_i]$. According to Lemma 4.5, $\Lambda_{t+1}[x_{i+1}] \geq \frac{\varepsilon}{2}(1-\beta)OPT_{t+1}[x_i]$ holds. Because $\Lambda_{t+1}[x_{i+1}]$ is not deleted at time $t+1$, 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 time $t+1$.

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 time $t+1$; otherwise, Condition 3 holds.

Therefore, at least one condition in Lemma 4.6 holds in all possible cases at time $t+1$, and we conclude the proof. □

Leveraging Lemma 4.6, we can analyze SIC theoretically. We then formally present the approximation guarantee and complexity of SIC in Theorems 4.7 and 4.8.

Theorem 4.7. *SIC maintains a $\frac{\varepsilon}{2}(1-\beta)$-approximate solution for SIM in $\Lambda_t[x_1]$ if 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 prove $\frac{\varepsilon}{2}(1-\beta)$ is a lower bound for the ratio between $\Lambda_t[x_1]$ and OPT_t. Let $\Lambda_t[x_0]$ be the expired checkpoint 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 4.6 holds at time t. If Condition 3 in Lemma 4.6 holds, then we have $OPT_t \le \varepsilon\Lambda_t[x_1]$, since $\Lambda_t[x_1]$ directly maintains an approximate solution w.r.t. W_t. Otherwise, we have $OPT_t \le OPT_t[x_0] \le \frac{2}{\varepsilon(1-\beta)}\Lambda_t[x_1]$, since $\Lambda_t[x_0]$ has expired. Thus, SIC maintains an $\frac{\varepsilon}{2}(1-\beta)$-approximate solution for SIM in $\Lambda_t[x_1]$. □

Theorem 4.8. *The number of checkpoints in SIC w.r.t. a sliding window of size N is $O(\frac{\log N}{\beta})$.*

Proof. Lemma 4.6 guarantees either $\Lambda_t[x_{i+1}]$ or $\Lambda_t[x_{i+2}]$ is less than $(1-\beta)\Lambda_t[x_i]$. Since $\frac{\Lambda_t[x_1]}{\Lambda_t[N]}$ is $O(N)$, 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 in 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 for each update is $O(\frac{dg\log N}{\beta})$. When SieveStreaming is used as the checkpoint oracle, SIC obtains a $(1/4-\beta)$-approximate solution for SIM. The time complexity of SIC for each update is $O(\frac{d\log N\log k}{\beta^2})$ accordingly.

Discussion. Although we have discussed how to handle SIM queries over sliding windows that slide for one action at a time, retrieving the result at such an intense rate is often unnecessary. Here, we discuss how to handle the case where each window slide receives L actions while the earliest L actions expire at the same time. For SIC, we create only one checkpoint when the window slides from W_t to W_{t+L}. Subsequently, we use all actions from a_{t+1} to a_{t+L} to update all checkpoints in the window. Finally, we adopt the same strategy as presented in Algorithm 1 to delete the checkpoints. The theoretical results of SIC are not affected by the above adaptations.

5 LOCATION-BASED SIC

As SIC does not consider any spatial information of social actions, it cannot be used to answer LSIM queries directly. In this section, we will show how SIC can be integrated with a well-known spatial index, i.e., Quadtree [42], to process LSIM queries efficiently.

Next, we first briefly present the procedure of Quadtree maintenance over sliding windows. Then, we will propose *Location-based SIC* (LSIC) framework and its improved version LSIC$^+$, in Sections 5.1 and 5.2, respectively, on processing continuous as well as ad hoc LSIM queries.

Quadtree Maintenance. Quadtree is a commonly used in-memory index for two-dimensional point data. In Quadtree, each internal node has exactly 4 quadrants that divide its minimum bounding rectangle (MBR) into four equal parts, namely, North West (NW), North East (NE), South West (SW), and South East (SE). Each quadrant corresponds to one child of the node. For ease of presentation, we encode the Quadtree nodes as follows: (1) the quadrants of any node are numbered as NW=0, NE=1, SW=2, and SE=3; (2) the children use the id of their parent as prefix, adding the number of its quadrant as the last digit. To maintain two-dimensional point data (i.e., the positions of actions in the geo-tagged social stream) in Quadtree, four basic operations are needed: (1) to insert a point into Quadtree; (2) to delete a point from Quadtree; (3) to split a Quadtree node; and (4) to combine Quadtree nodes into one node. The insertion and deletion of a point traverses Quadtree along a path from root to leaf and visits all Quadtree nodes whose MBRs cover the

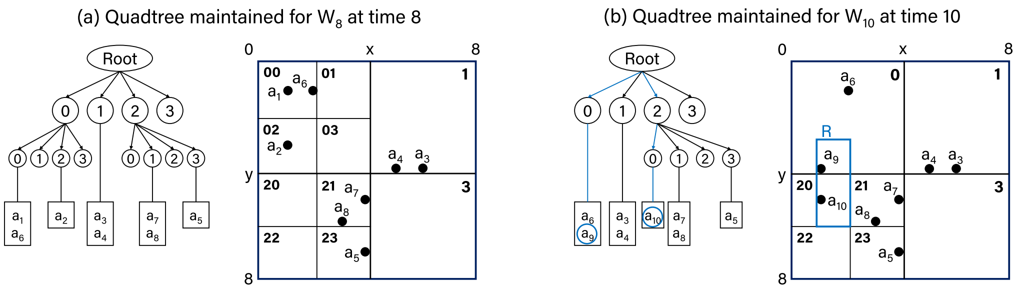

(a) Quadtree maintained for W8 at time 8
Root
0 1 2 3
a1 a6
a2
a3 a4
a7 a8
a5
00 01 02 03 20 21 22 23 1 3
(b) Quadtree maintained for W10 at time 10
a6 a9
a10
R
0 x 8 y

Fig. 3. The Quadtree index maintained at times 8 and 10 (node capacity $M = 2$ and fill factor $\alpha = 1.0$).

point. The point is finally inserted into or deleted from the point list in the leaf node. For query efficiency, Quadtree has a node capacity $M \in \mathbb{Z}^+$ that limits the maximum number of points in a leaf node. When a leaf node has contained more than M points after insertion, it will be *split* into four children corresponding to the quadrants of its MBR. Conversely, to improve the space usage, when an internal node has fewer points than a threshold, its descendant nodes will be *combined* into one node. Let $\alpha \in [0, 1]$ be the filling factor. When an internal node contains fewer than αM points, the *combination* procedure deletes its descendants from Quadtree and inserts all points in the deleted nodes into the node. The reason why a lower threshold is used for *combination* is to avoid frequent split/combination fluctuations, which incur excessive maintenance costs.

For LSIM processing, LSIC and LSIC$^+$ maintain a Quadtree to index all active actions in W_t at time t according to their geo-locations. For each window slide, L arrival actions are inserted into Quadtree while L expired ones are deleted from Quadtree (when $t > N$). One common operation on Quadtree for LSIM processing is the *region query* that retrieves the set of actions in a specified query region. Starting from the root node, a region query recursively traverses all nodes overlapped with the query region. When traversing an internal node, it further traverses its children overlapped with the query region. When traversing a leaf node, it iterates through the point list in the node and adds all actions covered by the query region into the result set. Finally, it collects the actions from each overlapped node as the result set for the region query.

Example 5.1. In Figure 3, we continue with Figure 1 to illustrate how Quadtree is maintained over a geo-tagged social action stream. At time 8, Quadtree maintains all active actions in W_8. Nodes 0 and 2 are split as more than $M = 2$ actions have been inserted into both nodes. When the window slides from $t = 8$ to $t = 10$, a_1 and a_2 are deleted while a_9 and a_{10} are inserted. The children of Node 0 are combined, because there is only two actions remaining in Node 0.

Next, we show how to process a region query in $R = [(1, 3), (2, 6)]$ with Quadtree. Starting from *Root*, we first check whether the MBRs of *Root*'s children are overlapped with R. We find Nodes 0 and 2 are overlapped with R. Node 0 is a leaf node and its point list will be traversed. a_9 is located in R but a_6 is not. Thus, a_9 is added to the result set. Node 2 is an internal node, and we further traverse its child Node 20 that are overlapped with R. Finally, it adds a_{10} in Node 20 to the result set and returns $\{a_9, a_{10}\}$ as the result for the region query.

5.1 The LSIC Framework

In this subsection, we propose the *Location-based Sparse Influential Checkpoints* (LSIC) framework for processing both continuous and ad hoc LSIM queries, but in different ways. Generally, the LSIC framework comprises two components: (1) a Quadtree index for actions in the current window;

(2) SIC instances to process LSIM queries. LSIC maintains an SIC instance for each continuous LSIM query and registers the instance in the Quadtree. Whenever an arrival action is inserted into the query region of the continuous LSIM, the SIC instance is updated accordingly. The SIC instance can always provide an approximate solution for the continuous LSIM at any time. However, such a method cannot work for ad hoc LSIM query processing. This is because ad hoc LSIM may be issued at any time in arbitrary regions and thus it is infeasible to maintain the seed sets for any possible queries in advance. To provide the solutions for any ad hoc LSIM queries in real time, LSIC maintains SIC instances in Quadtree nodes. When receiving an ad hoc LSIM query, LSIC first finds the SIC instance from the node that are covered by the query region and contains the maximum number of active actions as a partial solution. Then, LSIC feeds the remaining actions in the query region to the partial solution for the final solution. Next, we will discuss the detailed procedures of continuous and ad hoc LSIM processing with LSIC.

Continuous LSIM Processing with LSIC. We show how to process a continuous LSIM query $q_{[t_1,t_2]}(k,R)$ from time t_1 to t_2 with LSIC. For any time $t \in [t_1,t_2]$, LSIC should maintain a seed set for LSIM w.r.t. the actions located in R from a_{t_1} to a_t. The method of LSIC to process continuous LSIM naturally extends from SIC: it maintains an SIC instance to track the seed set for each continuous LSIM over time. The difference is that the SIC instance is only updated for actions in the query region of LSIM. The Quadtree is used to filter the actions that are irrelevant to the query. The operations to process an LSIM query $q_{[t_1,t_2]}(k,R)$ during time t_1 to t_2 are listed as follows.

- **Registration.** When $q_{[t_1,t_2]}(k,R)$ is received at time t_1, LSIC creates an empty SIC instance SIC_q and registers it in the Quadtree. Specifically, SIC_q is registered in the minimum node whose MBR can fully cover the query region R. We use *cur* to denote the node containing SIC_q in the Quadtree.
- **Update.** If an action a_t is inserted into Node *cur*, then it will check whether $\mathbf{p}_t$ is located in R. If $\mathbf{p}_t \in R$, then SIC_q will process a_t as an appending action according to Algorithm 1.
- **Movement.** The instance SIC_q may be moved to another node when Node *cur* is split or deleted. First, if *cur* is split and one of *cur*'s child can also fully cover R, SIC_q will be moved to that child. When none of *cur*'s children can fully cover R, SIC_q will not be moved. Second, if *cur* is deleted for combination, SIC_q will be moved back to the parent of *cur*.
- **Deletion.** When $q_{[t_1,t_2]}(k,R)$ expires at time t_2, SIC_q is terminated and deleted from *cur*.

At any time t from t_1 to t_2, LSIC can always maintain the solution of LSIM $q_{[t_1,t_2]}(k,R)$ in the SIC instance SIC_q.

Example 5.2. In Figure 4, we give an example for continuous LSIM query processing with LSIC. Two continuous queries $Q_1 = q_{[1,10]}(2,R_1)$ where $R_1 = [(3,5),(4,6)]$ and $Q_2 = q_{[1,10]}(2,R_2)$ where $R_2 = [(5,2),(8,4)]$ are received at time 1. They are initially inserted into *Root*. At time 3, *Root* is split. Q_1 and Q_2 are moved to Node 2 and Node 1, respectively. At time 8, Node 2 is split and Q_1 is moved to Node 21. At time 10, Nodes 00, 01, 02, 03 are combined back to Node 0. Since Q_1 and Q_2 are not registered in these nodes, they are not affected. During the period, Q_1 receives two actions a_7, a_8 and SIC_1 is updated for them. Q_2 receives a_3, a_4 and SIC_2 is also updated accordingly. At time 10, the seed set for Q_1 and Q_2 returned by SIC_1 and SIC_2 are $\{u_5\}$ and $\{u_3\}$, respectively.

Theoretical Analysis of LSIC for continuous LSIM. We give the approximation ratio of LSIC for continuous LSIM in Theorem 5.3.

THEOREM 5.3. *The result provided by LSIC for any continuous LSIM query is $\frac{\varepsilon}{2}(1-\beta)$-approximate when an ε-approximate checkpoint oracle is used.*

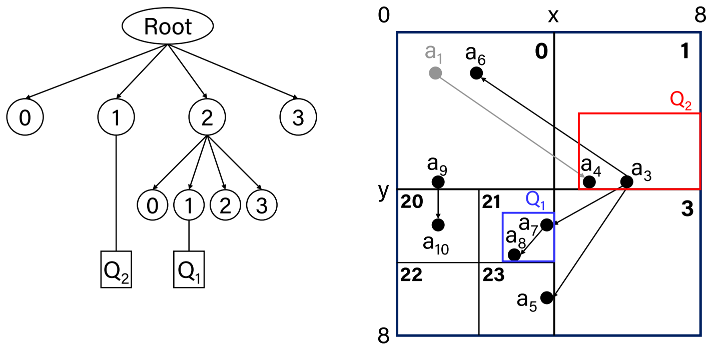


Fig. 4. Example for processing continuous LSIM queries Q_1 and Q_2 with LSIC. Note that Q_1 and Q_2 are registered in Quadtree and maintained with SIC instances SIC_1 and SIC_2, respectively.

Proof. At any time $t \in [t_1, t_2]$, a continuous LSIM query $q_{[t_1,t_2]}(k, R)$ maintains a seed set with the maximum influence value for actions located in R from a_{t_1} to a_t. Therefore, the theorem holds if (1) SIC_q maintained by LSIC processes a set of actions $\{a_i | i \in [t_1, t] \wedge \mathbf{p}_i \in R\}$ at time t; (2) SIC_q provides an $\frac{\varepsilon}{2}(1-\beta)$-approximate solution for the processed actions. Succinctly examining the operations, Condition (1) must hold for SIC_q. In addition, Condition (2) holds as the influence function $f(I_t(\cdot, R))$ for Region R preserves the monotonicity and submodularity of $f(I_t(\cdot))$. Therefore, SIC_q provides an $\frac{\varepsilon}{2}(1-\beta)$-approximate solution for LSIM $q_{[t_1,t_2]}(k, R)$ at any time $t \in [t_1, t_2]$. □

Next, we analyze the complexities of the above operations. The number of traversed nodes for the insertion of SIC_q is at most equal to the height of Quadtree, i.e., $O(\log \frac{N}{M})$. This is because Quadtree nodes in the same level are not overlapped with each other. Therefore, for any region R, there is at most one node can fully cover R. The time complexity for SIC_q to update an action a_t is $O(\frac{dg \log N_R}{\beta})$ where N_R is the number of active actions in R. The movements of SIC_q for the split and deletion of *cur* only involve *cur* and the parent/children of *cur*. Hence, the time complexity of any movement is $O(1)$. The deletion of SIC_q at t_2 has the same complexity as insertion, i.e., $O(\log \frac{N}{M})$. Finally, since the solution is always explicitly maintained in SIC_q, the time complexity to answer any query is $O(1)$.

Ad hoc LSIM Processing with LSIC. Next, we will present how to answer an ad hoc LSIM query $q_t(k, R)$ at time t using LSIC. When the query is received at time t, LSIC should return a seed set with the maximum influence value w.r.t. all actions that are (1) located in R and (2) active in W_t in real time. Compared to continuous LSIM, ad hoc LSIM are more flexible: a query may be issued at any time in an arbitrary region. Therefore, it is infeasible to precompute a seed set for any possible ad hoc queries. To answer ad hoc LSIM queries in real time, the LSIC framework maintains SIC instances in Quadtree nodes (an SIC instance in a Quadtree node processes all active actions in this node) and utilizes them to accelerate ad hoc LSIM processing. Although an SIC instance in a Quadtree node cannot provide a solution with a bounded approximation ratio for an ad hoc LSIM query, it can be used to provide a partial solution for LSIM whose query region can cover this node. As the partial solution has processed a large portion of actions in the query region, LSIC only queries for unprocessed actions and feeds them to the partial solution for the final solution. To provide the partial solutions for as many LSIM queries as possible, LSIC should maintain an SIC instance in every Quadtree node. But such a maintenance strategy is very costly and unnecessary. First, the insertion of any action will trigger the updates of the SIC instances maintained in all nodes along the insertion path and incurs excessive update costs. Second, the

ALGORITHM 2: Ad hoc LSIM Processing with LSIC

```
   Input: A LSIM query $q_t(k, R)$, a Quadtree
   Output: The solution for $q_t(k, R)$
1  Initialize an empty queue $QUEUE$ and enqueue Quadtree. $Root$ into $QUEUE$;
2  $nd \leftarrow nil, max \leftarrow 0$;
3  while $QUEUE$ is not empty do
4      Dequeue the first node in $QUEUE$ as $cur$;
5      if $R_{cur} \subseteq R$ and $num(cur) > max$ and there exists an SIC instance in node cur then
6          $nd \leftarrow cur, max \leftarrow num(cur)$;
7      else if $R_{cur} \not\subset R$ then
8          foreach child $cur.ch$ of node $cur$ do enqueue $cur.ch$ into $QUEUE$ if $R_{cur.ch} \cap R \neq nil$;
9  if $nd \neq nil$ then
10     Use $SIC_{nd}$ as the partial solution $sol$;
11     $A_r \leftarrow \{a_i | a_i \in W_t \wedge \mathbf{p}_i \in R \setminus R_{nd}\}$;
12 else
13     Invoke an empty SIC instance $sol$;
14     $A_r \leftarrow \{a_i | a_i \in W_t \wedge \mathbf{p}_i \in R\}$;
15 foreach $a$ in $A_r$ do $sol$.process($a$);
16 return the solution from $sol$;
```

space costs of maintaining SIC instances for all nodes are prohibitive. Third, the SIC instances for nodes with very small areas or a few actions hardly bring benefits to query processing. In practice, LSIC maintains an SIC instance in a node only if: (1) it contains at least M actions and (2) the area of its MBR exceeds a threshold (e.g., 0.4% of the full space). When an LSIM cannot find any SIC instance to provide the partial result, LSIC will perform a region query on Quadtree to retrieve all actions in the query region and process these actions using SSM to acquire the final solution.

The detailed description of processing an ad hoc LSIM query $q_t(k, R)$ with LSIC is shown in Algorithm 2. Given a Quadtree node cur, we use R_{cur} and $num(cur)$ to denote the MBR of cur and the number of actions in cur, respectively. LSIC first finds a Quadtree node nd such that (1) it maintains an SIC instance and (2) its MBR R_{nd} is fully covered by R, i.e., $R_{nd} \subseteq R$. When there is more than one node found, LSIC will select the node containing the maximum number of actions as nd (Lines 3–8). Then, LSIC considers two cases: (1) If nd exists, then the result of SIC_{nd} will be the partial result sol. The remaining actions A_r for $q_t(k, R)$ will include all active actions in Region $R \setminus R_{nd}$, i.e., $\{a_i | a_i \in W_t \wedge \mathbf{p}_i \in R \setminus R_{nd}\}$ (Lines 9–11). (2) If none of the nodes with SIC instances is fully covered by R, then LSIC will set up an empty instance sol and add all active actions in R into A_r (Lines 12–14). Finally, sol processes each action in A_r and is used to provide the final solution for $q_t(k, R)$ (Lines 15 and 16).

Example 5.4. In Figure 5, we give an example for ad hoc LSIM processing using LSIC. The Quadtree used is the same as Example 5.2 in Figure 4. We consider an SIC instance is maintained for every Quadtree node containing at least 2 actions (except the *Root* node). The nodes that only contain 0 or 1 action are not maintained with SIC instances. Given an LSIM query $Q_3 = q_{10}(2, R_3)$ where $R_3 = [(2, 3), (4, 8)]$, two Quadtree nodes, namely 21 and 23, are covered by R_3. As Node 23 does not maintain an SIC instance, the SIC instance in Node 21 provides the partial result sol. Then, we query the remaining actions that are not processed by sol yet. We have $A_r = \{a_5\}$ and sol will process a_5. $\{u_7\}$ is returned as the final solution of Q_3 after processing a_5. For an LSIM query $Q_4 = q_{10}(2, R_4)$ where $R_4 = [(4, 3), (7, 5)]$, we do not find any Quadtree node covered by R_4.

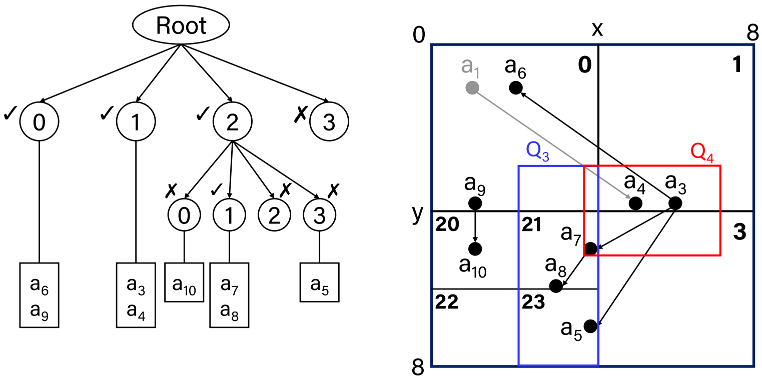


Fig. 5. Example for processing ad hoc LSIM queries Q_3 and Q_4 with LSIC. Note that we mark "✓" for Quadtree nodes with SIC instances and "✗" for Quadtree nodes without SIC instances.

Therefore, LSIC performs a region query of R_4 in Quadtree. The result is $A_r = \{a_3, a_4, a_7\}$. Then, the actions in A_r are processed from scratch according to SSM. The solution for Q_4 is $\{u_3\}$.

Theoretical Analysis of LSIC for ad hoc LSIM. The approximation ratio of the result returned by Algorithm 2 is given in Theorem 5.5.

THEOREM 5.5. *The solution returned by Algorithm 2 is $\frac{\varepsilon}{2}(1-\beta)$-approximate for any ad hoc LSIM query when an ε-approximate checkpoint oracle is used.*

PROOF. To prove Theorem 5.5, the first thing to note is that the approximation ratios of checkpoint oracles listed in Table 2 are order-independent. They do not make any assumption of the arrival order of actions and their results are ε-approximate even for the most adversary streaming order. SIC preserves the order-independence of checkpoint oracles. Then, we can prove Theorem 5.5 by showing that Algorithm 2 merely changes the order to process actions, which does not affect the approximation ratio. We consider two cases to process an ad hoc LSIM $q_t(k, R)$ in Algorithm 2.

Case 1 ($nd = nil$). In this case, LSIC retrieves all active actions in R and processes them from scratch. The procedure is equivalent to process SIM using the Set-Stream Mapping interface. From Theorem 4.1, we can see the result is an ε-approximate solution for $q_t(k, R)$.

Case 2 ($nd \neq nil$). In this case, we divide the active actions in R into two subsets: (1) $A_c = \{a_i | a_i \in W_t \wedge \mathbf{p}_i \in R_{nd}\}$; (2) $A_r = \{a_i | a_i \in W_t \wedge \mathbf{p}_i \in R \setminus R_{nd}\}$. Actions in A_c have been processed by SIC_{nd} and actions in A_r will be processed by the partial result of SIC_{nd}. As SIC_{nd} is maintained by Algorithm 2 over actions in A_c, one of three conditions in Lemma 4.6 must hold at time t. Then, actions in A_r are processed as appending actions. According to Lemma 4.6, one of three conditions can still hold after processing actions in A_r. Then, all actions in $A_c \cup A_r$ have been processed. Because $A_c \cup A_r$ have contained all actions required for $q_t(k, R)$ and SIC is order-independent, it is guaranteed that the final solution is $\frac{\varepsilon}{2}(1-\beta)$-approximate according to Theorem 4.7.

In both cases, Algorithm 2 provides an at least $\frac{\varepsilon}{2}(1-\beta)$-approximate solution for any ad hoc LSIM query and we conclude the proof. □

To analyze the complexity of ad hoc LSIM query processing with LSIC, we should consider both the maintenance of SIC instances in Quadtree nodes for each action and the cost of Algorithm 2 to return the final result. Inserting an action a_t into the Quadtree needs to traverse $O(\log \frac{N}{M})$ nodes from root to leaf. For each node, if it maintains an SIC instance, the instance will process

a_t using Algorithm 1. Therefore, the complexity of maintaining SIC instances in Quadtree nodes for each action is $O(\frac{dg \log N \log \frac{N}{M}}{\beta})$. Then, we analyze the complexity of Algorithm 2 for each ad hoc LSIM query. First, it traverses each Quadtree node overlapped with R to find the partial result. The number of nodes traversed is $O(\frac{N}{M})$ in the worst case. Then, the region query retrieves at most N_R actions to be processed by the partial result, where N_R is the number of active actions in R. If $nd \neq nil$, then the number of actions processed by the partial result will be $(N_R - N_{R_{nd}})$; otherwise, it will process N_R actions. Therefore, the complexity to process the partial result is $O(dgN_R)$. Finally, the complexity of Algorithm 2 to process an ad hoc LSIM query is $O(\frac{N}{M} + dgN_R)$.

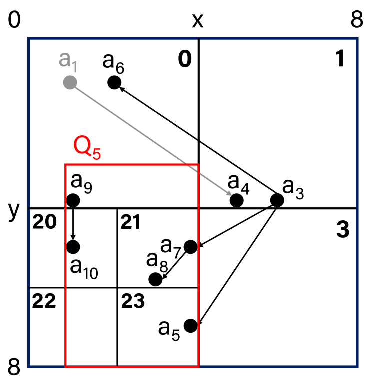
5.2 The LSIC$^+$ Framework

In this subsection, we introduce the LSIC$^+$ framework to improve upon LSIC. Under certain circumstances, the seed sets returned by LSIC show inferior quality compared with the state-of-the-art static IM approaches, e.g., IMM [44] and Greedy [35]. Therefore, we are motivated to improve LSIC so that LSIM can still be processed in real time while the seed quality can match the state-of-the-art static IM approaches.

At a high level, LSIC$^+$ maintains the same indices as LSIC: (1) a Quadtree spatial index for actions in the sliding window W_t at time t and (2) the SIC instances in Quadtree nodes, as introduced in Section 5.1. However, LSIC$^+$ employs a different accessing method to the maintained indices for LSIM processing. The basic idea is as follows: given a query region R, it acquires top-k' users with the maximum region-constrained influence value $f(I_t(\cdot, R))$. Then, it runs a simple Thresholding algorithm adapted from Reference [4] to retrieve the seed set for LSIM from the top-k' users. It is noted that $k' \geq k$ is a tunable parameter. A greater k' means better seed quality yet lower query efficiency and vice versa. Subsequently, we first describe the method to process continuous and ad hoc LSIM queries with LSIC$^+$, including (1) how to acquire the top-k' users in the query region R of an continuous or ad hoc LSIM and (2) how the Thresholding algorithm retrieves the seed set from the given top-k' users. Then, we analyze LSIC$^+$ theoretically.

LSIM processing with LSIC$^+$. Given a continuous LSIM $q_{[t_1,t_2]}(k, R)$, LSIC$^+$ also registers it in the Quadtree. The following operations: *Registration*, *Movement*, and *Deletion*, are the same as LSIC in Section 5.1. Instead of maintaining an SIC instance for each continuous LSIM, LSIC$^+$ maintains the influence value $f(I_t(u, R))$ of each user $u \in U$ in the query region R from time t_1 to t_2 and always tracks the top-k' users with the maximum influence values. Initially, the region-constrained influence set $I_t(u, R)$ of each user $u \in U$ is set to $\emptyset$ at time t_1. Then, for each window slide, LSIC$^+$ updates $I_t(u, R)$ as well as the influence value $f(I_t(u, R))$ of each user u w.r.t. the arrival and expiry of actions in R. Moreover, the top-k' influential users as well as their influence values in R are explicitly kept. Once there are any change in the top-k' users (both the entries of new users and the updates in existing users), LSIC$^+$ reruns Thresholding to update the seed set for $q_{[t_1,t_2]}(k, R)$. Given an ad hoc LSIM $q_t(k, R)$ at time t, LSIC$^+$ first runs a region query to acquire all Quadtree nodes that (1) are overlapped with R and (2) have SIC instances. Then, for every SIC instance in these nodes, LSIC$^+$ retrieves each user u from its solution and computes the influence value $f(I_t(u, R))$. Additionally, it obtains the influence values of remaining users directly from the actions in R. It combines both results and acquires the top-k' users with the maximum influence values in R for $q_t(k, R)$. Similarly, LSIC$^+$ runs Thresholding to obtain the solution for $q_t(k, R)$.

Given a set S' of top-k' users obtained for any LSIM query $q_t(k, R)$, the Thresholding algorithm to retrieve the seed set is presented in Algorithm 3. First, it initializes a priority queue PRQ and inserts each user u in S' to PRQ (Line 1). The candidate result $CAND$ is initialized to $\emptyset$. The users in PRQ are ordered by the marginal gain $u.\delta$ w.r.t. $CAND$ in descending order. The initial marginal gain $u.\delta$ of u is set to its influence value $f(I_t(u, R))$, because $CAND = \emptyset$. Then, it



Iteration (1): $TH = 2$

User	$I_{10}(u, R_5)$	δ
$\boldsymbol{u_3}$	$\boldsymbol{u_4, u_5}$	**2**
u_5	u_4, u_5	2
u_2	u_2, u_6	2
u_4	u_4	1
u_6	u_6	1
u_1	$\emptyset$	0

$CAND = \{u_3\}$

Iteration (1): $TH = 2$

User	$I_{10}(u, R_5)$	δ
$\boldsymbol{u_2}$	$\boldsymbol{u_2, u_6}$	**2**
u_4	u_4	1
u_5	u_4, u_5	0
u_6	u_6	1
u_1	$\emptyset$	0

$CAND = \{u_2, u_3\}$

Fig. 6. Example for processing an LSIM query Q_5 with Thresholding in LSIC$^+$.

ALGORITHM 3: Thresholding

```
   Input: A LSIM query $q_t(k, R)$, a set $S'$ of $k'$ users, a parameter $\gamma$
   Output: The solution for $q_t(k, R)$
1  Initialize a priority queue (max-heap) $PRQ$,insert each $u \in S'$ into $PRQ$ ordered by $u.\delta = f(I_t(u, R))$;
2  $TH \leftarrow \max_{u \in S'} f(I_t(u, R)), CAND \leftarrow \emptyset$;
3  while $PRQ$ is not empty and $|CAND| < k$ do
4      Peek the first user $u$ in $PRQ$;
5      while $TH \leq u.\delta$ do
6          Remove the first user $u$ from $PRQ$, update $u.\delta \leftarrow f(I_t(CAND \cup \{u\}, R)) - f(I_t(CAND, R))$;
7          if $TH \leq u.\delta$ and $|CAND| < k$ then
8              $CAND \leftarrow CAND \cup \{u\}$, if $|CAND| = k$ then break;
9          else
10             Insert $u$ into $PRQ$ if $u.\delta > 0$;
11         Peek the first user $u$ in $PRQ$;
12     $TH \leftarrow (1 - \gamma) \cdot TH$;
13 return $CAND$;
```

updates the candidate result $CAND$ in an iterative manner with a threshold TH. TH is initially set to the maximum influence value among the users in S' (Line 2). The procedure of an iteration with threshold TH is shown in Lines 3–12. It evaluates each user u whose marginal gain $u.\delta$ potentially reaches TH at the iteration. When $u.\delta$ of the first user u in PRQ is at least TH, it removes u from PRQ and re-evaluates its marginal gain $u.\delta$ w.r.t. $CAND$ (Line 6). If $u.\delta$ can still reach TH, then u will be added to $CAND$; otherwise, u will be inserted back to PRQ as long as $u.\delta > 0$ (Lines 7–10). If $CAND$ has contained k users, then no more iteration is needed and $CAND$ will be returned (Line 8). Otherwise, the threshold TH will be descended by $(1 - \gamma)$ times ($\gamma > 0$) for the next iteration (Line 12). Finally, $CAND$ will be returned as the seed set for $q_t(k, R)$ if either $CAND$ contains k elements or PRQ is empty (Line 13).

Example 5.6. In Figure 6, we give an example for LSIM processing using Thresholding in LSIC$^+$. We consider a query $Q_5 = q_{10}(2, R_5)$ where $R_5 = [(1, 3), (4, 8)]$ at time 10. Let $k' = 4$ and $\gamma = 0.1$. As a preliminary step, LSIC$^+$ retrieves four users $\{u_2, u_3, u_4, u_5\}$ with the maximum influence values for Thresholding. Initially, the threshold is set to $TH = 2$ and the candidate result is $CAND = \emptyset$. At Iteration (1) with $TH = 2$, it evaluates the first user u_3 and the marginal gain of adding u_3 to $CAND$ is 2. Therefore, u_3 is added to $CAND$. Then, u_5 is evaluated but the marginal gain of adding u_5 becomes 0 as $I_{10}(u_5, R_5)$ is identical to $I_{10}(u_3, R_5)$ and u_3 has been added to $CAND$. As a result, u_5 is not inserted to the priority queue any more. Next, u_2 is evaluated and included into $CAND$. Finally, as S has contained 2 users, no more iteration is required and $CAND = \{u_2, u_3\}$ is returned as the solution for Q_5.

Theoretical Analysis of LSIC$^+$. First, we give the approximation ratio of Thresholding for LSIM in Theorem 5.7 when we assume that the seed set is only selected from the top-k' users.

Theorem 5.7. *Given a set $S' \subseteq U$ of k' users with the maximum influence value $f(I_t(u, R))$, the solution $CAND$ returned by Algorithm 3 satisfies that $f(I_t(CAND, R)) \geq (1 - 1/e - \gamma)OPT'$, where $OPT' = \max_{S \subseteq S' : |S| \leq k} f(I_t(S, R))$.*

Proof. First, if $CAND$ has less than k elements, it must hold that $u.\delta = 0$ for each $u \in CAND \setminus S'$. We have $f(I_t(CAND, R)) = OPT'$ in this case, because adding any remaining users into $CAND$ cannot increase the influence value any more.

Then, we consider the case where $CAND$ contains k users. Let $S_j = \{u_1, \ldots, u_j\}$ ($j \in [1, k]$) be the first j users added to $CAND$ and $S_0 = \varnothing$. Assume that u_{j+1} is added to S_j at the iteration with a threshold TH. It holds that $u_{j+1}.\delta = f(I_t(S_j \cup \{u_{j+1}\}, R)) - f(I_t(S_j, R)) \geq TH$ and $u.\delta < \frac{TH}{1-\gamma}$ for any $u \in S' \setminus (S_j \cup \{e_{j+1}\})$. Then, $u_{j+1}.\delta \geq (1-\gamma)u.\delta$ for any $u \in S^* \setminus S_j$ where $S^* = \text{argmax}_{S \subseteq S' : |S| \leq k} f(I_t(S, R))$. By summing up the above inequality for each $u \in S^* \setminus S_j$, we have $|S^* \setminus S_j| \cdot u_{j+1}.\delta \geq (1-\gamma) \sum_{u \in S^* \setminus S_j} u.\delta$. Thus, $u_{j+1}.\delta \geq \frac{1-\gamma}{|S^* \setminus S_j|} \cdot \sum_{u \in S^* \setminus S_j} u.\delta \geq \frac{1-\gamma}{k} \cdot \sum_{u \in S^* \setminus S_j} u.\delta$. In addition, $\sum_{u \in S^* \setminus S_j} u.\delta \geq OPT' - f(I_t(S_j, R))$ for submodularity. Therefore, $u_{j+1}.\delta = f(I_t(S_{j+1}, R)) - f(I_t(S_j, R)) \geq \frac{1-\gamma}{k}(OPT' - f(I_t(S_j, R)))$. Equivalently, $f(I_t(S_{j+1}, R)) - OPT' \geq (1 - \frac{1-\gamma}{k})(f(I_t(S_j, R)) - OPT')$. Substituting S_{j+1} by $S_k, \ldots, S_1$ for k times, we prove $f(I_t(CAND, R)) = f(I_t(S_k, R)) \geq (1 - (1 - \frac{1-\gamma}{k})^k) \cdot OPT' \geq (1 - e^{-(1-\gamma)})OPT' \geq (1 - 1/e - \gamma)OPT'$. □

Given the conclusion of Theorem 5.7, it is obvious that LSIC$^+$ will return $(1 - 1/e - \gamma)$-approximate solution for any LSIM query if $k' = |U|$, i.e., all users in the social network are considered by Thresholding. However, this incurs a prohibitive computational cost for processing an LSIM in real time. In real-world scenarios, due to the power-law nature of social influence, i.e., only a few users achieve high influences and most users have very low influences, ignoring the users with low influences does not severely affect the solution quality. Therefore, we set k' to be slightly greater than k (e.g., $k' = 3k$ in the experiments). Although there is no theoretical guarantee of the approximation factor of LSIC$^+$ unless $k' = |U|$, it can still return solutions of good quality for LSIM empirically even when k' is small. The complexity of LSIC$^+$ is analyzed as follows. First, it maintains the region-constrained influence values as well as the top-k' users for each continuous LSIM. An arrival or expiry of any action triggers updates in at most d influence sets. In addition, we consider the top-k' users are maintained in a red-black tree and thus updating the top-k' users for any action is $O(d \log k')$. For any ad hoc LSIM, LSIC$^+$ traverses $O(\frac{N}{M})$ nodes in Quadtree. The complexity of acquiring the top-k' users is $O(dN_R \log k')$. Finally, the Thresholding algorithm runs at most $O(\frac{\log N_R}{\gamma})$ iterations, because the ratio between the influence values of the first user and the k'th user is $O(N_R)$. At each iteration, it evaluates at most k' users and takes $O(\log k')$ to remove a user from or insert a user to a priority queue. Therefore, the complexity of Algorithm 3 is $O(\frac{k' \log N_R}{\gamma}(\log k' + g))$.

Table 3. Statistics of Datasets

Dataset	Users	Actions	Average Depth	Geo-tagged
Reddit	2,628,904	48,104,875	4.58	N
Twitter	2,881,154	9,724,908	1.87	N
Twitter-SG	177,934	1,830,086	1.22	Y
Twitter-NY	1,196,237	13,267,644	1.38	Y

6 EXPERIMENTAL RESULTS

In this section, we evaluate the efficiency and effectiveness of our proposed frameworks on several real-world datasets. We first describe the experimental setup in Section 6.1. We then compare SIC with the state-of-the-art static and dynamic IM algorithms for SIM in Section 6.2. Finally, we evaluate the efficiency and effectiveness of LSIC and LSIC$^+$ for LSIM in Section 6.3.

6.1 Experimental Setup

Datasets. The following four real-world datasets are used in the experiments.

- **Reddit.** Reddit is an online forum where user actions include *post* and *comment*. We download the Reddit *comments* in May 2015 from *kaggle*[7] and query the Reddit API for the *posts* in the same period. We combine *posts* with *comments* and sort all actions by timestamp.
- **Twitter.** Twitter is an online social network where user actions include *tweet*, *retweet*, *quote* and *reply*. We collect these actions for one week via Twitter stream API[8] on trending topics such as 2016 US presidential election, 2016 NBA finals and UEFA Euro 2016.
- **Twitter-SG.** We collect geo-tagged user actions in Singapore via Twitter stream API. The bounding box is set to $[(1.2, 103.5), (1.5, 104.1)]$.
- **Twitter-NY.** We collect geo-tagged user actions in New York City via Twitter stream API. The bounding box is set to $[(40.5, -74), (40.9, -73)]$.

In data preprocessing, the geo-positions of actions in the **Twitter-SG** and **Twitter-NY** datasets are normalized to $[0, 1]^2$.

The statistics of these datasets are summarized in Table 3.

Approaches. The compared approaches are listed as follows.

- **IMM [44].** 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 w.r.t. W_t as directed edges. For location-aware IM, we construct a location-based influence graph $G_{t,R}$ for R only considering the *influences* to users in R w.r.t. W_t. The edge probabilities are assigned by the Weighted Cascade model (WCM) [22]. To extract the influential users, we set the parameters to $\varepsilon = 0.5, l = 1$ [44] and run IMM on the generated influence graphs.
- **UBI [10].** We use the state-of-the-art method Upper Bound Interchange (UBI) for IM on dynamic graphs as another baseline. The generation of influence graphs is the same as IMM. Then, a sequence of influence graphs $\{G_1, \dots, G_m\}$ are fed to UBI in a chronological order to track the influential users. We keep the same interchange threshold as used in Reference [10], i.e., 0.01. As UBI cannot be trivially adapted to location-aware IM and our results in

[7]https://www.kaggle.com/reddit/reddit-comments-may-2015.
[8]https://developer.twitter.com/en/docs.

Section 6.2 have shown the superiority of SIC over UBI, we do not consider UBI any more in Section 6.3.

- **GR [35].** We also implement the classic Greedy (GR) algorithm in Reference [35], because it achieves the best possible approximation factor of $(1 - 1/e)$ for SIM and LSIM queries. A detailed description of the algorithm is presented in Section 4.1. Since Greedy does not store any intermediate result, it always reruns from scratch when being queried. For SIM and LSIM processing, the only difference is the computation of influence values. Given a query region R, LSIM only considers active actions in R to compute the influence values.
- **IC.** The Influential Checkpoints (IC) framework for SIM is simply presented in Section 4.1 and a more detailed description can be found in Reference [50]. SieveStreaming [3] is used as the checkpoint oracle for IC, which guarantees $(1/2 - \beta)$-approximate solutions for SIM.
- **SIC.** The Sparse Influential Checkpoints (SIC) framework for SIM is presented in Section 4.2. SieveStreaming [3] is also used as the checkpoint oracle for SIC, which guarantees $(1/4 - \beta)$-approximate solutions for SIM. In Section 6.3, we use SIC as a baseline method for LSIM. SIC tracks a seed set over the stream without considering the spatial information of actions.
- **LSIC.** The LSIC framework for LSIM is proposed in Section 5.1. SieveStreaming [3] is still used as the checkpoint oracle for LSIC. LSIC returns $(1/4 - \beta)$-approximate solutions for both continuous and ad hoc LSIM.
- **LSIC$^+$.** The LSIC$^+$ framework for LSIM is proposed in Section 5.2. LSIC$^+$ could return $(1 - 1/e - \gamma)$-approximate solutions for continuous and ad hoc LSIM if k' were set to $|U|$. In practice, we set $k' = 3k$ and $\gamma = 0.1$ across all the experiments in Section 6.3.

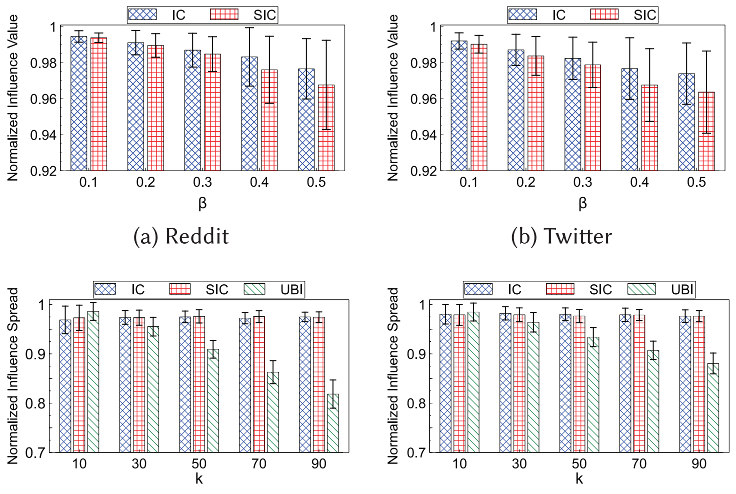
Existing location-aware IM methods [26, 28] are based on static graphs where both network topology and users' positions are not changed over time. Therefore, they cannot support incremental updates over social action streams and need to recompute the solution from scratch for each update. In our scenario, their performance cannot exceed IMM. Hence, they are not compared with our proposed approaches in the experiments.

Metrics for Solution Quality. We use two different metrics to evaluate the seed quality of compared approaches, because IMM and UBI work under the Independent Cascade model, whereas Greedy, IC, SIC, LSIC, and LSIC$^+$ are proposed for SIM and LSIM queries in Section 3. Given a seed set, (1) its **influence value** is the value of influence functions for SIM and LSIM (i.e., the cardinality functions) in Section 3; (2) its **influence spread** is the expected number of users activated by the seeds on the influence graph under the Independent Cascade model. We run 10,000 rounds of Monte-Carlo simulations to calculate the *influence spread*. It is a commonly used metric for the quality of IM algorithms [22]. After obtaining the *influence values* and *influence spread* of the seed sets returned by compared approaches, we further normalize them for ease of presentation. The *influence values* of all approaches are normalized by Greedy. For example, given an SIM or LSIM query, if the influence values of the seed sets returned by SIC and Greedy are 95 and 100, respectively, SIC will acquire a score of 0.95. Similarly, the *influence spread* is normalized by IMM. After the normalization, we use the *average scores* and *standard variances* to evaluate the solution quality of compared approaches. The normalized scores of *influence values* and *influence spread* are referred to as **Normalized Influence Value** and **Normalized Influence Spread**.

Metrics for Efficiency. For SIM and continuous LSIM processing, we use **throughput** as the metric for efficiency. The *throughput* is the average number of actions processed by an approach per second. A higher *throughput* means a faster speed for stream processing. We do not use the *average latency* as a metric, because the seeds are always maintained explicitly and the time to retrieve them is negligible. For ad hoc LSIM processing, we use both the **throughput** to maintain

Table 4. Parameters in the Experiments

Parameter	Values
β	0.1, **0.2**, 0.3, 0.4, 0.5
k	10, 30, **50**, 70, 90
N	$100K$, $300K$, **500K**, $700K$, $900K$
Selectivity	1%, **2%**, 3%, 4%, 5%
Q	10, 30, **50**, 70, 90



(a) Reddit (b) Twitter

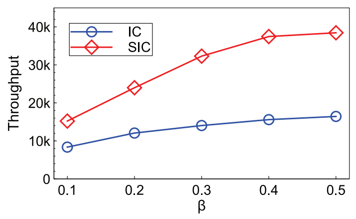
(c) Reddit (d) Twitter

Fig. 7. Solution quality of compared approaches for SIM with varying parameter β and seed size k. The average score is represented by the bar length and the standard variance is denoted by the error bar length. Note that the influence values in Figures (a) and (b) are normalized by Greedy and the influence spread in Figures (c) and (d) are normalized by IMM (the same for Figures 9 and 11).

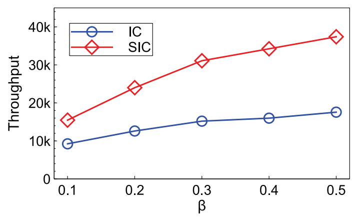
SIC instances in Quadtree nodes over streams and the **average latency**, which is the average time from receiving an ad hoc query to returning its solution as the metrics for efficiency.

Query Workload of LSIM. The query workload of continuous LSIM is as follows: We first generate Q LSIM queries. At the start of the stream, we submit Q queries to compared approaches. The results of Q queries are requested for each window slide until the end of the stream. The query workload of ad hoc LSIM is as follows: we generate 1, 000 LSIM queries and assign a random timestamp to each of them. During the stream, each LSIM query will be issued to compared approaches at the assigned timestamp.

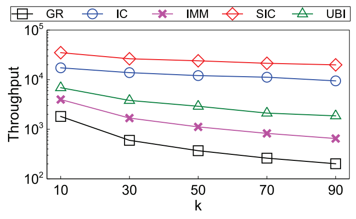
Parameters. We examine five parameters in the experiments: (1) β is the parameter in IC and SIC to achieve a trade-off between solution quality and efficiency. For LSIC, we set $\beta = 0.1$ across all the experiments. (2) k is the maximum size of the seed set. (3) N is the size of the sliding window. (4) *Selectivity* is the ratio between the area of the query region for LSIM and the full space. The area of the full space has been normalized to 1 in data preprocessing. If the query region is set to a square of side 0.1, then the *Selectivity* will be 1%. (5) Q is the number of continuous LSIM queries monitored at the same time. We fix the number of actions for each window slide to $L = 5{,}000$. The summary of parameters is listed in Table 4 with default values in bold.



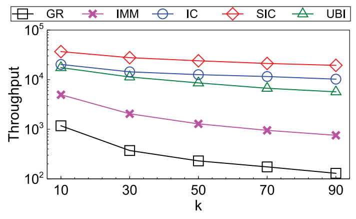
(a) Reddit



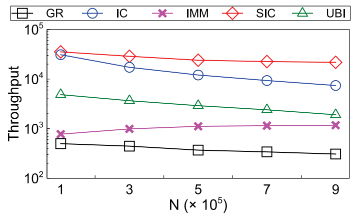
(b) Twitter



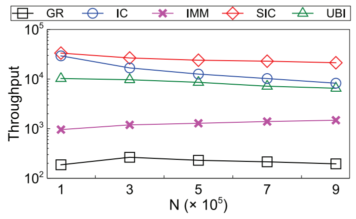
(c) Reddit



(d) Twitter



(e) Reddit



(f) Twitter

Fig. 8. Throughputs of compared approaches for SIM with varying β, seed size k, and window size N.

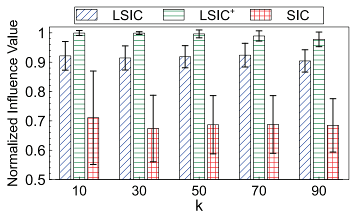
Experiment Settings. All experiments are conducted on a server running Ubuntu 16.04.3 LTS. It has an Intel Xeon E7-4820 1.9GHz processor and 128GB memory. All compared approaches except IMM are implemented in Java 8. The IMM implementation available[9] is written in C++.

6.2 Evaluation for SIM

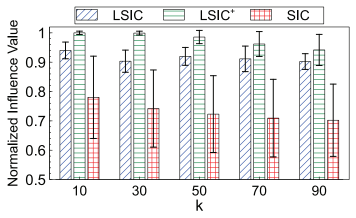
Solution Quality for SIM. In Figures 7(a) and 7(b), we test the *influence values* of IC and SIC with varying β. The average scores for the influence values of SIC are slightly lower than IC while the standard variances are a little higher in most of the experiments, because fewer checkpoints are maintained by SIC. However, both SIC and IC show good robustness against β in terms of solution quality. The influence values of IC and SIC are still over 96% of the influence values of Greedy when $\beta = 0.5$.

The results for *influence spread* with varying k are presented in Figures 7(c) and 7(d). Compared with IMM, IC and SIC show less than 5% quality losses in all experiments. This result verifies the effectiveness of SIM, because the seeds for SIM queries achieve nearly equivalent influence spread to the seeds retrieved by IMM under the Independent Cascade model. In contrast, although the influence spread of UBI is close to IMM when $k = 10$, it degrades dramatically when k increases. UBI relies on interchanging users to maintain the seeds against the updates on the influence graph. It interchanges a user into the seed set only when a substantial gain is achieved in the estimated

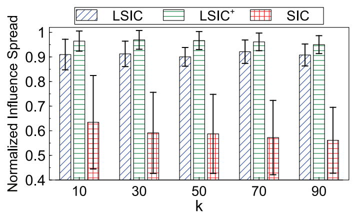
[9]https://sourceforge.net/projects/im-imm/.



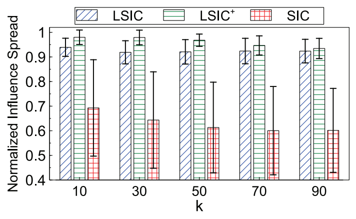
(a) Twitter-SG



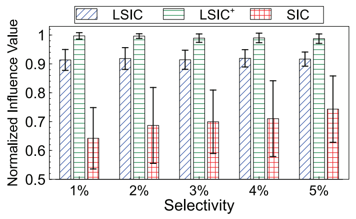
(b) Twitter-NY



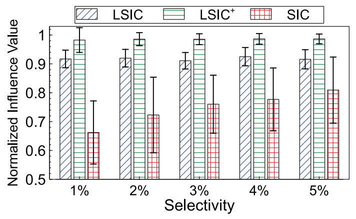
(c) Twitter-SG



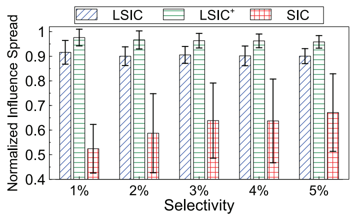
(d) Twitter-NY



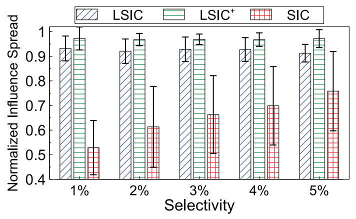
(e) Twitter-SG



(f) Twitter-NY



(g) Twitter-SG

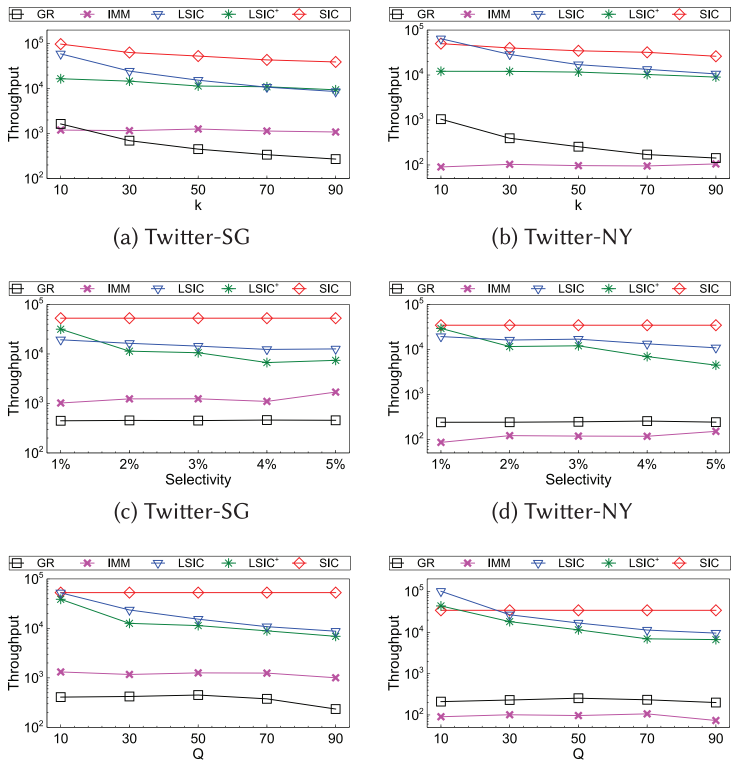


(h) Twitter-NY

Fig. 9. Solution quality of compared approaches for continuous LSIM with varying k and selectivity. The average score and the standard variance are denoted by the length of the bar and the error bar, respectively.

influence spread (i.e., 1% of the total influence spread prior to the interchange). As the total influence spread increases with k, users with emerging influences are harder to be interchanged into the seed set. As a result, it delays the interchanges and degrades the solution quality.

Efficiency for SIM. The *throughputs* of IC and SIC with varying β are presented in Figures 8(a) and 8(b). Both approaches achieve higher throughputs when β increases. There are two reasons behind such an observation. First, SieveStreaming is used as the checkpoint oracle and its update cost is inversely proportional to β. Second, SIC maintains fewer checkpoints for a larger β (note that the number of checkpoints in IC is not affected by β), which naturally leads to shorter update time. Therefore, SIC shows greater superiority over IC in terms of throughput when β increases.



(a) Twitter-SG (b) Twitter-NY

(c) Twitter-SG (d) Twitter-NY

(e) Twitter-SG (f) Twitter-NY

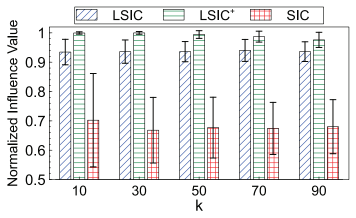
Fig. 10. Throughputs of compared approaches for continuous LSIM with varying k, selectivity, and Q.

The *throughputs* of compared approaches with varying k are presented in Figures 8(c) and 8(d). The throughputs of all approaches are inversely correlated with k. IC and SIC employ SieveStreaming as the checkpoint oracle, whose update time is proportional to $O(\log k)$. Hence, the throughputs of IC and SIC drop when k is larger. Compared with IC and the baseline methods, SIC shows significant advantages in throughputs for all experiments. SIC dominates Greedy and IMM by achieving up to two orders of magnitude higher throughputs in both datasets. The throughputs of UBI are far behind SIC and IC on the *Reddit* dataset but are close to IC on the *Twitter* dataset. Nonetheless, SIC still has more than three times higher throughputs than UBI.

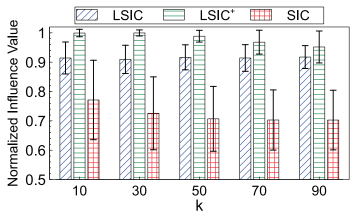
The *throughputs* of compared approaches with varying N are presented in Figures 8(e) and 8(f). Although the throughputs of all approaches decrease with increasing N, SIC shows better scalability than other approaches, since it only maintains $O(\log N)$ checkpoints when β is fixed. In addition, IC shows a nearly equal throughput to SIC when $N = 100K$. This is because the number of checkpoints in IC are very close to SIC (20 in IC vs. 12 in SIC) and the benefits of sparse checkpoints become less significant. Nonetheless, when N increases, SIC regains its superiority. In addition, SIC achieves throughputs of up to 178×, 35×, and 12× over Greedy, IMM, and UBI, respectively.

6.3 Evaluation for LSIM

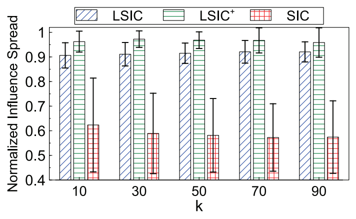
Solution Quality for Continuous LSIM. The *solution quality* of compared approaches for continuous LSIM queries is shown in Figures 9(a)–9(h). First, the average scores of LSIC for both



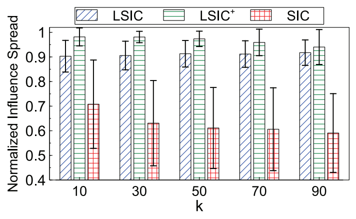
(a) Twitter-SG



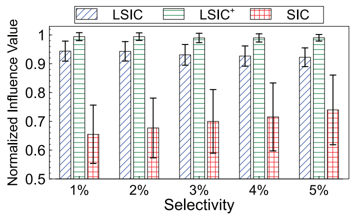
(b) Twitter-NY



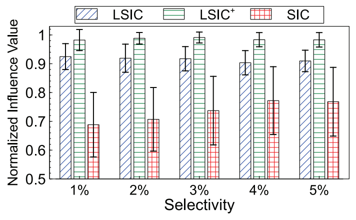
(c) Twitter-SG



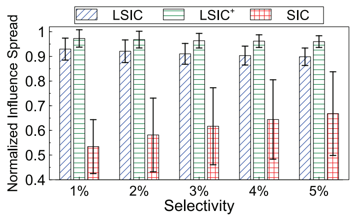
(d) Twitter-NY



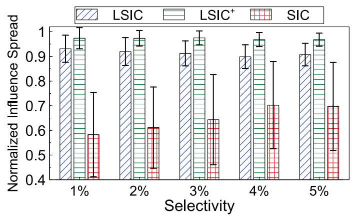
(e) Twitter-SG



(f) Twitter-NY



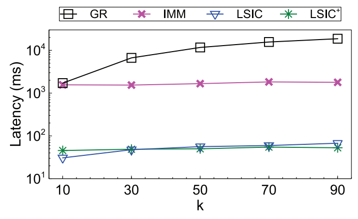
(g) Twitter-SG



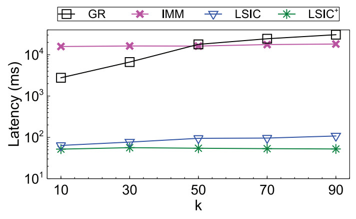
(h) Twitter-NY

Fig. 11. Solution quality of compared approaches for ad hoc LSIM with varying k and selectivity. The average score is represented by the bar length and the standard variance is denoted by the error bar length.

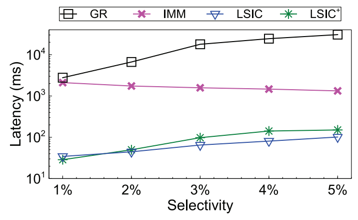
quality metrics are over 90% in all experiments while the standard variances are at most 6.5%. The results support our argument on the effectiveness of LSIC for continuous LSIM. However, the seed quality of LSIC is 5%–10% lower than Greedy and IMM. The main reason is that, compared with SIM, the optimal influence value of an LSIM query is obviously smaller. Both SIC and LSIC employ SieveStreaming as the checkpoint oracle while SieveStreaming depends on estimating the optimal influence value accurately to maintain near-optimal solutions for SIM and LSIM. Therefore, smaller optimal influence values cause larger relative errors in estimations and degrade the solution quality of LSIC for LSIM compared with SIC for SIM. Second, LSIC$^+$ significantly improves the seed quality upon LSIC. LSIC$^+$ achieves average scores of 94%–99% in all experiments with smaller variances. Due to the power-law nature of social influence, the quality losses caused by



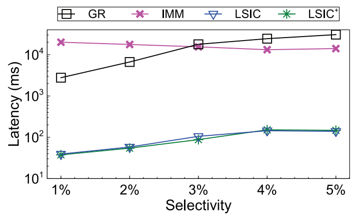
(a) Twitter-SG



(b) Twitter-NY



(c) Twitter-SG



(d) Twitter-NY

Fig. 12. Average latency of compared approaches for ad hoc LSIM with varying k and selectivity.

ignoring the users with fewer influence values are marginal. The seed quality of LSIC$^+$ is nearly equivalent to Greedy and IMM when k is 10 or 30. But the seed quality slightly degrades when k increases. Nevertheless, the average scores are always above 94% in all experiments. Finally, SIC cannot provide solutions of good quality for LSIM, because the spatial information is totally ignored. The average scores of SIC are mostly between 50% and 80%. But the average score of SIC obviously increases with *selectivity*, since global and local influencers tend to be more similar when the selectivity increases.

Efficiency for Continuous LSIM. The *throughputs* of compared approaches for continuous LSIM with varying k, *selectivity* and Q are shown in Figures 10(a)–10(f). Overall, the throughputs of LSIC and LSIC$^+$ are at least $8K$ and $5K$, respectively. Both of them achieve up to two orders of magnitude higher throughput than Greedy and IMM. This results confirm the high efficiency of both approaches for continuous LSIM processing. LSIC achieves higher throughputs than LSIC$^+$ in most cases, especially when k is small or the selectivity is large. Furthermore, the throughputs of LSIC and LSIC$^+$ decrease with increasing k, *selectivity*, and Q. Such a trend is apparent, because the total number of influence function evaluations for each window slide naturally grows with these parameters. The throughputs of Greedy are significantly lower on the *Twitter-NY* dataset, because *Twitter-NY* contains about eight times more users than *Twitter-SG*. Different from other approaches, the throughputs of IMM increase with rising selectivities because a smaller selectivity often leads to sparser influence graphs, where IMM runs more slowly [44]. In spite of this, IMM still has much lower throughputs than LSIC and LSIC$^+$ when the selectivity is 5%. Finally, although SIC achieves the highest throughputs among compared approaches, we have shown that it cannot provide solutions of good quality for LSIM in Figures 9(a)–9(h).

Solution Quality for Ad Hoc LSIM. The *solution quality* of compared approaches for ad hoc LSIM queries with varying k and *selectivity* is presented in Figures 11(a)–11(h). The results are generally similar to those of continuous LSIM. The average scores of LSIC and LSIC$^+$ are at least 89% and 94%, respectively. LSIC$^+$ also shows a slight degradation in solution quality when k increases. In addition, the solution quality of LSIC for ad hoc LSIM drops when the *selectivity* increases. This is because ad hoc LSIM queries with smaller selectivities are less likely to find a partial solution.

These queries are handled by rerunning from scratch, which often returns solutions with slightly better quality, though this incurs higher computational costs. Not surprisingly, SIC also cannot provide high-quality solutions for ad hoc LSIM.

Efficiency for Ad Hoc LSIM. The *average latency* of compared approaches for ad hoc LSIM queries with varying k and *selectivity* are presented in Figures 12(a)–12(d). Generally, the average latencies of LSIC and LSIC$^+$ are both within 150ms, which is acceptable for real-time query processing over streams. Greedy and IMM often take 1 to 20s for an ad hoc LSIM query. Compared with them, LSIC and LSIC$^+$ achieve speedups of up to 512× and 476× for ad hoc LSIM processing. The average latencies of LSIC and LSIC$^+$ are quite close. LSIC is more scalable to the larger selectivity thanks to the partial solutions maintained in Quadtree nodes. LSIC$^+$ shows better scalability to k, because it only considers top-k' users. In addition, the *throughputs* of LSIC and LSIC$^+$ to update SIC instances in Quadtree nodes are still over 10K. The throughput is about 18K when $k = 50$ and 12K when $k = 90$. The results confirm LSIC and LSIC$^+$ are able to maintain the indices and answer ad hoc LSIM queries at the same time.

7 CONCLUSION

In this article, we proposed the *Stream Influence Maximization* (SIM) and *Location-aware SIM* (LSIM) queries to retrieve a seed set with the maximum influence value over location-unaware and location-aware social streams, respectively. Then, we proposed the *Sparse Influential Checkpoints* (SIC) framework to process SIM queries efficiently over social streams. Theoretically, SIC could return $\frac{\varepsilon}{2}(1-\beta)$-approximate solutions for SIM queries. Furthermore, we proposed the *Location-based SIC* (LSIC) framework and its improved version LSIC$^+$ for efficient LSIM processing over geo-tagged social streams. LSIC also provided $\frac{\varepsilon}{2}(1-\beta)$-approximate solutions for both ad hoc and continuous LSIM queries. LSIC$^+$ improved the seed quality of LSIC and it would achieve a $(1-1/e-\gamma)$-approximation factor for LSIM if $k' = |U|$. Finally, extensive experiments on real-world datasets demonstrated that our proposed frameworks achieved up to two orders of magnitude speedups over the state-of-the-art IM approaches. Meanwhile, compared with them, the losses in seed quality were at most 4%, 11%, and 6% for SIC, LSIC, and LSIC$^+$, respectively. In particular, they achieved an average throughput of at least 5K actions per second, which indicated that they were adequate to process real-world social streams.

For future work, we consider that users have various preferences in the social media advertising domain [29, 32, 55]. We plan to study the streaming influence maximization query where advertisers can input keywords/topics to target the audience that is of interest to the query.

REFERENCES

[1] Charu C. Aggarwal, Shuyang Lin, and Philip S. Yu. 2012. On influential node discovery in dynamic social networks. In *Proceedings of the 2012 SIAM International Conference on Data Mining (SDM'12)*. SIAM, 636–647. DOI: http://dx.doi.org/10.1137/1.9781611972825.55

[2] Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and Vangelis Th. Paschos. 2012. Online maximum k-coverage. *Discrete Appl. Math.* 160, 13–14 (2012), 1901–1913. DOI: http://dx.doi.org/10.1016/j.dam.2012.04.005

[3] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*. ACM, 671–680. DOI: http://dx.doi.org/10.1145/2623330.2623637

[4] Ashwinkumar Badanidiyuru and Jan Vondrák. 2014. Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. SIAM, 1497–1514. DOI: http://dx.doi.org/10.1137/1.9781611973402.110

[5] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. 2014. Maximizing social influence in nearly optimal time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. SIAM, 946–957. DOI: http://dx.doi.org/10.1137/1.9781611973402.70

[6] Vladimir Braverman and Rafail Ostrovsky. 2007. Smooth histograms for sliding windows. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE Computer Society, 283–293. DOI: http://dx.doi.org/10.1109/FOCS.2007.55

[7] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. 2011. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. ACM, 665–674. DOI: http://dx.doi.org/10.1145/1963405.1963499

[8] Shuo Chen, Ju Fan, Guoliang Li, Jianhua Feng, Kian-Lee Tan, and Jinhui Tang. 2015. Online topic-aware influence maximization. *Proc. VLDB Endow.* 8, 6 (2015), 666–677. DOI: http://dx.doi.org/10.14778/2735703.2735706

[9] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*. ACM, 1029–1038. DOI: http://dx.doi.org/10.1145/1835804.1835934

[10] Xiaodong Chen, Guojie Song, Xinran He, and Kunqing Xie. 2015. On influential nodes tracking in dynamic social networks. In *Proceedings of the 2015 SIAM International Conference on Data Mining (SDM'15)*. SIAM, 613–621. DOI: http://dx.doi.org/10.1137/1.9781611974010.69

[11] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM J. Comput.* 31, 6 (2002), 1794–1813. DOI: http://dx.doi.org/10.1137/S0097539701398363

[12] Pedro M. Domingos and Matthew Richardson. 2001. Mining the network value of customers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*. ACM, 57–66. DOI: http://dx.doi.org/10.1145/502512.502525

[13] Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2017. Submodular optimization over sliding windows. In *Proceedings of the 26th International Conference on World Wide Web (WWW'17)*. ACM, 421–430. DOI: http://dx.doi.org/10.1145/3038912.3052699

[14] Ju Fan, Jiarong Qiu, Yuchen Li, Qingfei Meng, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, and Xiaoyong Du. 2018. OCTOPUS: An online topic-aware influence analysis system for social networks. In *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE'18)*. IEEE Computer Society, 1569–1572. DOI: http://dx.doi.org/10.1109/ICDE.2018.00178

[15] Uriel Feige. 1998. A threshold of ln n for approximating set cover. *J. ACM* 45, 4 (1998), 634–652. DOI: http://dx.doi.org/10.1145/285055.285059

[16] Manuel Gomez-Rodriguez and Bernhard Schölkopf. 2012. Influence maximization in continuous time diffusion networks. In *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*. Retrieved from http://icml.cc/2012/papers/189.pdf.

[17] Manuel Gomez-Rodriguez, Le Song, Nan Du, Hongyuan Zha, and Bernhard Schölkopf. 2016. Influence estimation and maximization in continuous-time diffusion networks. *ACM Trans. Inf. Syst.* 34, 2 (2016), 9:1–9:33. DOI: http://dx.doi.org/10.1145/2824253

[18] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. 2010. Learning influence probabilities in social networks. In *Proceedings of the 3rd International Conference on Web Search and Web Data Mining (WSDM'10)*. ACM, 241–250. DOI: http://dx.doi.org/10.1145/1718487.1718518

[19] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. 2011. A data-based approach to social influence maximization. *Proc. VLDB Endow.* 5, 1 (2011), 73–84. DOI: http://dx.doi.org/10.14778/2047485.2047492

[20] Long Guo, Dongxiang Zhang, Gao Cong, Wei Wu, and Kian-Lee Tan. 2017. Influence maximization in trajectory databases. *IEEE Trans. Knowl. Data Eng.* 29, 3 (2017), 627–641. DOI: http://dx.doi.org/10.1109/TKDE.2016.2621038

[21] Keke Huang, Sibo Wang, Glenn S. Bevilacqua, Xiaokui Xiao, and Laks V. S. Lakshmanan. 2017. Revisiting the stop-and-stare algorithms for influence maximization. *Proc. VLDB Endow.* 10, 9 (2017), 913–924. DOI: http://dx.doi.org/10.14778/3099622.3099623

[22] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*. ACM, 137–146. DOI: http://dx.doi.org/10.1145/956750.956769

[23] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. 2015. Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.* 2, 3 (2015), 14:1–14:22. DOI: http://dx.doi.org/10.1145/2809814

[24] Konstantin Kutzkov, Albert Bifet, Francesco Bonchi, and Aristides Gionis. 2013. STRIP: Stream learning of influence probabilities. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. ACM, 275–283. DOI: http://dx.doi.org/10.1145/2487575.2487657

[25] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. Van Briesen, and Natalie S. Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. ACM, 420–429. DOI: http://dx.doi.org/10.1145/1281192.1281239

[26] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-Lee Tan, and Wen-Syan Li. 2014. Efficient location-aware influence maximization. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*. ACM, 87–98. DOI: http://dx.doi.org/10.1145/2588555.2588561

[27] Hui Li, Sourav S. Bhowmick, Aixin Sun, and Jiangtao Cui. 2015. Conformity-aware influence maximization in online social networks. *VLDB J.* 24, 1 (2015), 117–141. DOI: http://dx.doi.org/10.1007/s00778-014-0366-x

[28] Xiao Li, Xiang Cheng, Sen Su, and Chenna Sun. 2018. Community-based seeds selection algorithm for location aware influence maximization. *Neurocomputing* 275 (2018), 1601–1613. DOI: http://dx.doi.org/10.1016/j.neucom.2017.10.007

[29] Yuchen Li, Zhifeng Bao, Guoliang Li, and Kian-Lee Tan. 2015. Real time personalized search on social networks. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE'15)*. IEEE Computer Society, 639–650. DOI: http://dx.doi.org/10.1109/ICDE.2015.7113321

[30] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence maximization on social graphs: A survey. *IEEE Trans. Knowl. Data Eng.* PP, 99 (2018). DOI: http://dx.doi.org/10.1109/TKDE.2018.2807843

[31] Yuchen Li, Ju Fan, Dongxiang Zhang, and Kian-Lee Tan. 2017. Discovering your selling points: Personalized social influential tags exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*. ACM, 619–634. DOI: http://dx.doi.org/10.1145/3035918.3035952

[32] Yuchen Li, Dongxiang Zhang, Ziquan Lan, and Kian-Lee Tan. 2016. Context-aware advertisement recommendation for high-speed social news feeding. In *Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE'16)*. IEEE Computer Society, 505–516. DOI: http://dx.doi.org/10.1109/ICDE.2016.7498266

[33] Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. 2015. Real-time targeted influence maximization for online advertisements. *Proc. VLDB Endow.* 8, 10 (2015), 1070–1081. DOI: http://dx.doi.org/10.14778/2794367.2794376

[34] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. 2011. Sparsification of influence networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*. ACM, 529–537. DOI: http://dx.doi.org/10.1145/2020408.2020492

[35] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.* 14, 1 (1978), 265–294. DOI: http://dx.doi.org/10.1007/BF01588971

[36] Hung T. Nguyen, Preetam Ghosh, Michael L. Mayo, and Thang N. Dinh. 2017. Social influence spectrum at scale: Near-optimal solutions for multiple budgets at once. *ACM Trans. Inf. Syst.* 36, 2 (2017), 14:1–14:26. DOI: http://dx.doi.org/10.1145/3086700

[37] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. 2016. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD'16)*. ACM, 695–710. DOI: http://dx.doi.org/10.1145/2882903.2915207

[38] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken ichi Kawarabayashi. 2016. Dynamic influence analysis in evolving networks. *Proc. VLDB Endow.* 9, 12 (2016), 1077–1088. DOI: http://dx.doi.org/10.14778/2994509.2994525

[39] Barna Saha and Lise Getoor. 2009. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proceedings of the 9th SIAM International Conference on Data Mining (SDM'09)*. SIAM, 697–708. DOI: http://dx.doi.org/10.1137/1.9781611972795.60

[40] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. 2008. Prediction of information diffusion probabilities for independent cascade model. In *Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part III (KES'08)*. Springer-Verlag, 67–75. DOI: http://dx.doi.org/10.1007/978-3-540-85567-5_9

[41] Muhammad Aamir Saleem, Rohit Kumar, Toon Calders, Xike Xie, and Torben Bach Pedersen. 2017. Location influence in location-based social networks. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM'17)*. ACM, 621–630. DOI: http://dx.doi.org/10.1145/3018661.3018705

[42] Hanan Samet. 1984. The quadtree and related hierarchical data structures. *ACM Comput. Surv.* 16, 2 (1984), 187–260. DOI: http://dx.doi.org/10.1145/356924.356930

[43] Karthik Subbian, Charu C. Aggarwal, and Jaideep Srivastava. 2016. Querying and tracking influencers in social streams. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM'16)*. ACM, 493–502. DOI: http://dx.doi.org/10.1145/2835776.2835788

[44] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*. ACM, 1539–1554. DOI: http://dx.doi.org/10.1145/2723372.2723734

[45] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*. ACM, 75–86. DOI: http://dx.doi.org/10.1145/2588555.2593670

[46] Ali Vardasbi, Heshaam Faili, and Masoud Asadpour. 2017. SWIM: Stepped weighted shell decomposition influence maximization for large-scale networks. *ACM Trans. Inf. Syst.* 36, 1 (2017), 6:1–6:33. DOI: http://dx.doi.org/10.1145/3072652

[47] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2016. Distance-aware influence maximization in geo-social network. In *Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE'16)*. IEEE Computer Society, 1–12. DOI: http://dx.doi.org/10.1109/ICDE.2016.7498224

[48] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2017. Efficient distance-aware influence maximization in geo-social networks. *IEEE Trans. Knowl. Data Eng.* 29, 3 (2017), 599–612. DOI: http://dx.doi.org/10.1109/TKDE.2016.2633472

[49] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Chen Chen. 2017. Bring order into the samples: A novel scalable method for influence maximization. *IEEE Trans. Knowl. Data Eng.* 29, 2 (2017), 243–256. DOI: http://dx.doi.org/10.1109/TKDE.2016.2624734

[50] Yanhao Wang, Qi Fan, Yuchen Li, and Kian-Lee Tan. 2017. Real-time influence maximization on dynamic social streams. *Proc. VLDB Endow.* 10, 7 (2017), 805–816. DOI: http://dx.doi.org/10.14778/3067421.3067429

[51] Yuan Wang, Dongxiang Zhang, Lu Hu, Yang Yang, and Loo Hay Lee. 2017. A data-driven and optimal bus scheduling model with time-dependent traffic and demand. *IEEE Trans. Intell. Transport. Syst.* 18, 9 (2017), 2443–2452. DOI: http://dx.doi.org/10.1109/TITS.2016.2644725

[52] Yu Yang, Zhefeng Wang, Jian Pei, and Enhong Chen. 2017. Tracking influential individuals in dynamic networks. *IEEE Trans. Knowl. Data Eng.* 29, 11 (2017), 2615–2628. DOI: http://dx.doi.org/10.1109/TKDE.2017.2734667

[53] Mao Ye, Xingjie Liu, and Wang-Chien Lee. 2012. Exploring social influence for recommendation: A generative model approach. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'12)*. ACM, 671–680. DOI: http://dx.doi.org/10.1145/2348283.2348373

[54] Dongxiang Zhang, Yuchen Li, Xin Cao, Jie Shao, and Heng Tao Shen. 2018. Augmented keyword search on spatial entity databases. *VLDB J.* 27, 2 (2018), 225–244. DOI: http://dx.doi.org/10.1007/s00778-018-0497-6

[55] Dongxiang Zhang, Yuchen Li, Ju Fan, Lianli Gao, Fumin Shen, and Heng Tao Shen. 2017. Processing long queries against short text: Top-k advertisement matching in news stream applications. *ACM Trans. Inf. Syst.* 35, 3 (2017), 28:1–28:27. DOI: http://dx.doi.org/10.1145/3052772

[56] Dongxiang Zhang, Dingyu Yang, Yuan Wang, Kian-Lee Tan, Jian Cao, and Heng Tao Shen. 2017. Distributed shortest path query processing on dynamic road networks. *VLDB J.* 26, 3 (2017), 399–419. DOI: http://dx.doi.org/10.1007/s00778-017-0457-6

[57] Tao Zhou, Jiuxin Cao, Bo Liu, Shuai Xu, Ziqing Zhu, and Junzhou Luo. 2015. Location-based influence maximization in social networks. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM'15)*. ACM, 1211–1220. DOI: http://dx.doi.org/10.1145/2806416.2806462

[58] Honglei Zhuang, Yihan Sun, Jie Tang, Jialin Zhang, and Xiaoming Sun. 2013. Influence maximization in dynamic social networks. In *Proceedings of the 2013 IEEE 13th International Conference on Data Mining (ICDM'13)*. IEEE Computer Society, 1313–1318. DOI: http://dx.doi.org/10.1109/ICDM.2013.145