Influence Persistence Maximization in Temporal Social Networks

Xueqin Chang
Zhejiang University
changxq@zju.edu.cn

Qing Liu
Zhejiang University
qingliucs@zju.edu.cn

Baihua Zheng
Singapore Management University
bhzheng@smu.edu.sg

Yunjun Gao
Zhejiang University
gaoyj@zju.edu.cn

Abstract

In this paper, we investigate a novel Influence Persistence Maximization (InfPM) problem in temporal social networks. Given a temporal graph, InfPM aims to identify a fixed seed node set S that maximizes the total duration of persistent influence across consecutive snapshots. After proving that InfPM is NP-hard, monotonic, and non-submodular, we develop two efficient solutions: (1) RevG, a reverse greedy algorithm that iteratively removes low-contribution nodes, and (2) LRep, a replacement-based method that progressively improves the quality of seed node set. To accelerate influence computation in RevG and LRep, we propose a new influence computation method integrating snapshot compression, probability-aware sampling, and a specialized influence estimator offering unbiased estimation. Additionally, we explore a practical variant of InfPM, termed Win-InfPM, which relaxes the requirement of consecutive snapshots by introducing a flexible time window model. Extensive experiments on seven real-world networks demonstrate that (1) RevG and LRep effectively identify high-quality seed nodes, achieving up to 100% improvement in total influence persistence over the optimal baseline; and (2) the proposed influence computation method improves the efficiency of RevG and LRep by up to 400%, while maintaining comparable influence persistence.

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at https://github.com/ZJU-DAILY/INFPM.

1 Introduction

Influence Maximization (IM) [22] is a fundamental problem in social network analysis. Given a graph G and a budget k, IM aims to select a set of k seed nodes that maximizes the expected number of influenced nodes under a specified diffusion model. IM has broad applications in areas such as marketing [13], social recommendation [53], and diffusion control [27].

Most existing work assumes static networks [15, 16, 21, 22, 40, 42], where user relationships and diffusion patterns remain unchanged over time. However, real-world social networks exhibit continuous evolution [25, 26]. For example, every minute, approximately 400 new users join Facebook and more than 510,000 comments are posted [14]. To better capture such dynamics, recent studies have explored IM in temporal social networks [50]. Existing works in this area can be clustered into two main categories: (1) One-time Influence Maximization: selecting a seed node set to influence the maximum number of nodes at a specific timestamp [30, 46, 47, 54, 55], and (2) Adaptive Influence Maximization: continuously updating the seed node set to maximize its influence as the network evolves [35, 39, 49, 51, 52]. However, these approaches suffer from two key limitations:

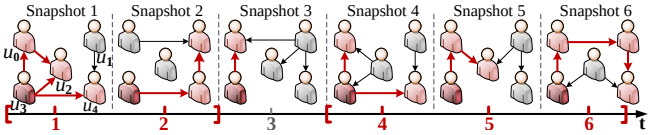


Figure 1: A motivation example with $S = \{u_3\}$

(1) **Short-lived Influence**: Targeting a single timestamp often leads to brief influence spikes with limited impact [23]. In practice, whether in marketing campaigns or public health initiatives, user adoption rarely occurs after a single exposure [6]. Successful outcomes - such as product adoption or behavioral change - typically require repeated exposures over time [12, 34].

(2) **Costly and Impractical Updates**: Continuously adjusting the seed node set can be infeasible due to rapid network changes and high deployment costs. For example, marketing campaigns involving top influencers may cost over $20K+ per iteration [38], which makes frequent updates of the seed node set prohibitively expensive.

The Influence Persistence Maximization Problem. To address these limitations, we propose the Influence Persistence Maximization (InfPM) problem in temporal social networks. Specifically, given a temporal graph $\mathcal{G}$, represented as a sequence of snapshots, a budget k, an influence threshold α, and a persistence threshold θ, InfPM aims to identify a *fixed* seed node set $S \subset V$ ($|S| = k$) that maximizes *influence persistence*. We define the influence persistence as the total duration of all time intervals that satisfy the following conditions: (1) each interval spans at least θ consecutive timestamps; and (2) in every snapshot within the interval, the seed node set S influences at least an α-fraction of nodes. Figure 1 illustrates InfPM through a six-snapshot temporal network. With parameters $k = 1$, $\alpha = 50\%$, and $\theta = 2$, a snapshot is considered sufficiently influenced when at least three nodes are influenced. As shown, if $S = \{u_3\}$, the intervals $[1, 2]$ and $[4, 6]$ satisfy the InfPM conditions. Thus, $S = \{u_3\}$ is the optimal solution, achieving a maximum influence persistence of 5. The InfPM problem has many real-life applications.

Application 1. *Research has shown that repeated product exposure over consecutive days significantly enhances customer familiarity and purchase likelihood [6, 34]. Consequently, when promoting new products on platforms such as Instagram and YouTube, companies tend to prioritize identifying influential users who can consistently reach and impact large audiences over time [24, 37]. In this context, InfPM helps identify users with sustained influence, improving campaign effectiveness and boosting conversion rates.*

Proceedings of the VLDB Endowment, Vol. 19, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Application 2. *Public health agencies such as WHO and CDC usually leverage sustained information diffusion for effective health interventions [2]. For instance, communities exposed to continuous vaccination messaging for at least a week often show significantly higher vaccine uptake rates compared to those receiving intermittent or short-term messaging [12]. InfPM can assist public health agencies in identifying individuals capable of maintaining long-term influence, thereby amplifying the reach and impact of health campaigns.*

Challenges and Solutions. We prove that the InfPM problem is **NP**-hard and its objective function is monotone but *non-submodular*. Traditional greedy methods [22] for Influence Maximization, which build the seed node set incrementally from an empty set, encounter a *cold-start issue* in the InfPM setting. Specifically, when the seed set is empty or very small, the influence persistence is often zero, since the influence threshold cannot be satisfied and no qualified time intervals can be found. Consequently, the influence persistence gain (i.e., the increase in influence persistence from adding a new node to the current seed set) also becomes zero, causing traditional greedy methods to fail to select effective seed nodes in the *initial stages*. To address this challenge, we propose two effective seed set selection algorithms: ReverseGreedy (RevG) and LazyReplace (LRep). Unlike traditional greedy approaches that build seed node sets incrementally from scratch, both RevG and LRep are designed to maintain meaningful influence persistence throughout the selection process, thus avoiding the cold-start problem.

- RevG works in a top-down manner. It starts with a well-constructed candidate seed node set and iteratively removes the nodes that contribute the least to influence persistence until the set is reduced to size k. To generate a high-quality candidate set, we apply a multi-step filtering process: (1) Identify the size-k seed node set that maximizes influence for each snapshot. (2) Retain only those snapshots where (i) the seed set influences at least α-fraction of nodes, and (ii) there are at least θ such snapshots in succession. (3) Aggregate all the seed node sets of snapshots from the second step to form the initial candidate set. RevG then removes nodes one at a time, selecting at each iteration the node whose removal minimizes the reduction in influence persistence - preserving as much influence persistence as possible.
- LRep implements a replacement-based optimization strategy. It starts with the top-k influential nodes, determined by their average influence spread across all snapshots, and iteratively refines this set by strategically replacing weaker seeds with stronger candidates to maximize influence persistence. To ensure computational efficiency, LRep uses a two-phase lazy evaluation strategy: (1) Rank candidate nodes based on their aggregated influence across all snapshots. (2) For each high-ranking candidate node n_c, first check whether replacing the weakest seed in the current seed set (i.e., the node contributing least to influence persistence) improves the overall influence persistence. If so, evaluate all nodes in the current seed set by considering their replacement with n_c and identify the one whose replacement yields the largest influence persistence. The identified seed node is then swapped with n_c. The process continues until no further beneficial replacement can be identified.

Both RevG and LRep require frequent computation of influence spread for the seed set, a task known to be #**P**-hard [22]. Existing methods like Monte Carlo simulations [22] and advanced reverse influence sampling (RIS) [22] are computational expensive in this setting. We develop a more efficient influence spread computation method with three key innovations. First, we introduce a snapshot compression technique that transforms each snapshot into a directed acyclic graph (DAG) through edge sampling and strongly connected component decomposition, which reduces graph size and shortens sampling paths. Second, we propose a novel sampling technique for reverse reachable sets generation on the compressed DAG, which assigns differential sampling probabilities to different components, rather than treating all nodes equally and sampling them uniformly as in the classical RIS method. Third, as the new sampling technique renders classical unbiased estimators ineffective, we design a specialized influence estimator tailored to the proposed sampling method to ensure unbiased estimation, with theoretical guarantees. For each snapshot graph G_t, the estimated influence spread lies within a factor of $(1-\epsilon)$ and $(1+\epsilon)\cdot(\prod_{j=1}^{\ell}\mathrm{Scr}(G_t[C_j]))^{-1}$ of the true value, where $\epsilon \in (0,1)$ and $\prod_{j=1}^{\ell}\mathrm{Scr}(G_t[C_j])$ denotes the overall strongly connected reliability of G_t.

We also explore Win-InfPM, a generalized variant of InfPM that relaxes the strict requirement of θ-consecutive-snapshots. In Win-InfPM, a time interval is qualified if each time window within it contains at least θ snapshots in which the seed set influences at least an α-fraction of nodes. To solve Win-InfPM efficiently, we adapt RevG and LRep by integrating a novel influence persistence computation algorithm PssW. This extension preserves the computational advantages of our original approaches while accommodating the more flexible temporal constraints of Win-InfPM.

Contributions. Our contributions are summarized as follows.

- We formally define the InfPM problem for the first time, and prove its NP-hardness through rigorous analysis.
- We propose two effective seed selection algorithms, RevG and LRep, and develop an efficient influence computation method with provably unbiased estimation.
- We introduce Win-InfPM, a practical extension that supports a flexible window-based influence persistence model.
- We conduct extensive experiments on seven real-world network, demonstrating the effectiveness, efficiency, and scalability of our proposed algorithms.

Roadmap. Section 2 reviews the related work. Section 3 formally defines and analyzes the InfPM problem. Section 4 presents RevG and LRep algorithms. Section 5 introduces influence computation algorithms. Section 6 presents the Win-InfPM variant and its solutions. Section 7 reports experimental results, and Section 8 concludes the paper.

2 Related Work

Influence Maximization in Static Social Networks. The Influence Maximization (IM) problem was first formulated as a discrete optimization problem by Kempe et al. [22], focusing on two fundamental propagation models: the independent cascade and linear threshold models. Although IM is **NP**-hard under both models, the monotonicity and submodularity of the objective function allow the use of greedy algorithms with a $(1-1/e)$-approximation guarantee. Since then, substantial follow-up research has focused on developing efficient and scalable IM solutions [16, 17, 19, 21, 40, 42, 43], with

many approaches leveraging reverse influence sampling (RIS) techniques [5]. In parallel, researchers have proposed a wide range of IM variants for specialized scenarios [4, 7–9, 18, 41, 45, 48]. Additionally, [1, 31, 32] provide comprehensive surveys and experimental evaluation of IM.

Influence Maximization in Temporal Social Networks. Recent research on influence maximization in temporal social networks, as reviewed in [50], falls into two main categories: (1) Snapshot-specific seed selection: These approaches select seed node sets to maximize influence at particular time instances [30, 46, 47, 54, 55]. For example, Wang et al. [46] modeled temporal networks as social streams and studied efficient seed selection over recent social actions. Zhao et al. [54, 55] proposed algorithms for seed selection at given query times while accounting for social activity decay. (2) Dynamic seed node set updating: Methods in this category focus on efficiently updating seed node sets to maximize influence as the network changes over time [35, 39, 49, 51, 52]. For instance, Ohsaka et al. [35] developed a fully dynamic scheme for real-time seed node set updates with node/edge changes. Yang et al. [51] tackled the problem of tracking top-k influential users in dynamic networks under a bounded relative error. Our work makes *two fundamental differences* from existing IM research. (1) Prior studies focus on maximizing influence at specific timestamps, whereas ours aims to maintain sufficient influence across multiple consecutive timestamps. (2) Some of previous approaches often update the seed node set dynamically, while ours seeks a single, fixed seed node set that maximizes influence persistence throughout the entire temporal network.

3 Preliminaries

In this section, we formally define the Influence Persistence Maximization problem and analyze its fundamental properties.

3.1 Problem Formulation

We model the temporal social network as a sequence of snapshot graphs $\mathcal{G} = (V, E) = \{G_t\}_{t=1}^{T} = \{G_1, G_2, ..., G_T\}$ ($t \in \mathbb{N}^+$) as in prior work [20, 29, 39]. We assume that the set of nodes remains fixed across all snapshot graphs and the edges in each snapshot graph change over different timestamps. Each snapshot graph $G_t(V, E_t) \in \mathcal{G}$ is modeled as a directed graph, where V and E_t represent the sets of $|V|$ nodes and $|E_t|$ edges, respectively. The edges over all snapshot graphs comprise $E = \bigcup_{t=1}^{T} E_t$. For a directed edge $e_t = (u, v) \in E_t$, we refer to u as an in-neighbor of v and v as an out-neighbor of u. Correspondingly, we employ $N_t^{in}(v)$ (resp. $N_t^{out}(v)$) to denote the set of in-neighbors (resp.out-neighbors) of v in G_t. Each directed edge $e_t = (u, v) \in E_t$ is associated with an influence weight $w_t(u, v) \in [0, 1]$, quantifying the influence of node u on node v at timestamp t. Note that the influence weights $w_t(u, v)$ over different timestamp t may be different.

In each snapshot graph, the influence diffuses independently. In this paper, we consider two basic and widely adopted diffusion models, i.e., the independent cascade (IC) and linear threshold (LT) models [22]. Given a snapshot graph G_t and a set of seed nodes S, both models simulate influence spread through an iterative process. At step 0, the seed nodes in S are activated and the other nodes are inactive. The IC and LT models differ in the following steps.

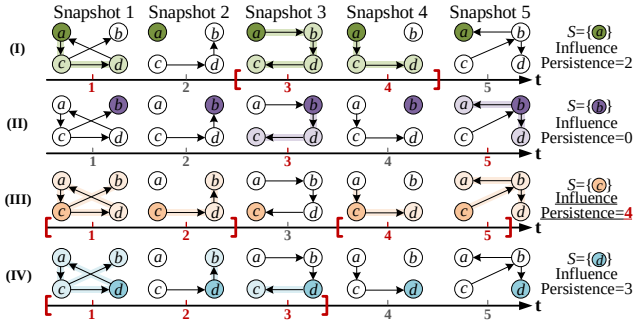


Figure 2: A simple temporal graph

Specifically, in the IC model, when a node u is activated at step $i \geq 0$, it attempts to activate each inactive out-neighbor v at step $i+1$ with probability $w_t(u, v)$, with only one attempt permitted. In the LT model, each node is associated with a uniformly sampled activation threshold $\rho_v^t \in [0, 1]$. At step $i > 0$, an inactive node v is activated iff the sum of influence weights from its activated in-neighbors exceeds ρ_v^t, i.e., $\sum_{u \in N_t^{in}(v) \wedge u \text{ is activated}} w_t(u, v) \geq \rho_v^t$. Noticeably, the LT model requires that each node v satisfies $\sum_{u \in N_t^{in}(v)} w_t(u, v) \leq 1$. In both models, activated nodes remain active, and the process terminates when no further activations occur, yielding the set of eventually activated nodes, denoted as $I(G_t, S)$. Accordingly, the expected influence spread of S in snapshot G_t is

$$\sigma_t(S) = \mathbb{E}[|I(G_t, S)|]. \tag{1}$$

Definition 1. **(Maximal (α, θ, S)-persistent influence interval).** *Given a temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, a seed node set $S \subset V$, and parameters $\alpha \in (0, 1]$ and $\theta \geq 1$ ($\theta \in \mathbb{N}^+$), a time interval $[t_s, t_e] \subseteq [1, T]$ is a maximal (α, θ, S)-persistent influence interval for $\mathcal{G}$ if the following conditions hold:*

1) ***α-validity:*** *$\forall t \in [t_s, t_e]$, $\sigma_t(S) \geq \alpha \cdot |V|$.*
2) ***θ-persistence:*** *$|t_e - t_s + 1| \geq \theta$.*
3) ***Maximalist:*** *$\nexists\, [t_s', t_e'] \supset [t_s, t_e]$ satisfies conditions 1) and 2).*

In other words, a maximal (α, θ, S)-persistent influence interval $[t_s, t_e]$ ensures that the expected influence spread of S meets the coverage threshold $\alpha \cdot |V|$ within the interval and that the interval spans at least θ consecutive timestamps. The interval is maximal in the sense that it cannot be extended further without violating these conditions. Note that multiple such intervals may exist for a given S. Consider the temporal graph $\mathcal{G}$ in Figure 2 (III), with $S = \{c\}$, $\alpha = 0.5$, and $\theta = 2$. The time intervals $[1, 2]$ and $[4, 5]$ are both valid maximal $(0.5, 2, \{c\})$-persistent influence intervals since: 1) at each timestamp in these intervals, $\sigma_t(\{c\}) \geq 0.5 \cdot 4 = 2$; 2) each interval has a length of 2; and 3) neither intervals can be extended while satisfying the above two conditions.

Definition 2. **(Influence persistence).** *Given a temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, a seed node set $S \subset V$, and two parameters $\alpha \in (0, 1]$ and $\theta \geq 1$, let $\mathcal{T}(S, \mathcal{G}) = \{[t_{s1}, t_{e1}], ..., [t_{sr}, t_{er}]\}$ be the set of all maximal (α, θ, S)-persistent influence intervals. The influence persistence of S in $\mathcal{G}$ w.r.t. α and θ, denoted by $F(\alpha, \theta, S)$, is:*

$$F(\alpha, \theta, S) \triangleq \begin{cases} \sum_{i=1}^{r} (t_{ei} - t_{si} + 1), & \mathcal{T} \neq \emptyset \\ 0, & \mathcal{T} = \emptyset \end{cases} \tag{2}$$

That is, the influence persistence of S in a temporal graph $\mathcal{G}$ equals to the total length of all maximal (α, θ, S)-persistent influence intervals. Continuing the example from Figure 2(III): we have

$\mathcal{T}(\{c\}, \mathcal{G}) = \{[1,2],[4,5]\}$, and thus $F(0.5, 2, \{c\}) = 2+2 = 4$. Building upon the above definitions, we formally define the InfPM problem.

Problem 1. **(InfPM).** *Given a temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, a budget k, and two parameters $\alpha \in (0,1]$ and $\theta \geq 1$, the objective of the InfPM problem is to find a seed node set $S \subset V$ with $|S| = k$ that maximizes influence persistence of S. Formally,*

$$S = \arg\max_{S \subset V \wedge |S|=k} F(\alpha, \theta, S). \tag{3}$$

Example 1. *Consider the temporal graph $\mathcal{G}$ with five snapshots in Figure 2. Let $k = 1$, $\alpha = 50\%$, and $\theta = 2$. Among all seed sets of size 1, selecting $S = \{c\}$ yields the longest total duration of maximal $(0.5, 2, \{c\})$-persistent influence intervals, namely $[1,2]$ and $[4,5]$, with $F(0.5, 2, \{c\}) = 4$. Hence, InfPM returns $S = \{c\}$.*

3.2 Problem Analyses

In this section, we first demonstrate that the InfPM problem is NP-hard. We further show that its influence persistence function is monotonic but not submodular under both the IC and LT models.

Theorem 1. *The InfPM problem is NP-hard.*

Proof. We prove this by reduction from the Decision version of the Set Cover (DSC) problem, which is known to be NP-complete. Given a ground set $U = \{u_1, u_2, \ldots, u_n\}$ and a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ where each $S_i \subseteq U$, the DSC problem asks whether there exist k subsets in $\mathcal{S}$ that together cover all elements in U. We construct an instance of InfPM as follows. Create a snapshot graph G_1 with two disjoint node sets: 1) $X = \{x_1, x_2, \cdots, x_m\}$, where each x_i corresponds to a subset $S_i \in \mathcal{S}$; 2) $Y = \{y_1, y_2, \cdots, y_n\}$, where each y_j corresponds to an element $U_j \in \mathcal{U}$. For each pair (S_i, u_j) such that $u_j \in S_i$, add a directed edge from x_i to y_j with weight 1. The temporal graph $\mathcal{G}$ consists of a single snapshot G_1. Next, we set the InfPM parameters: $\alpha = |U|/(|U| + |\mathcal{S}|)$, $\theta = 1$, and k (same as in the DSC). Assume a deterministic propagation model where each seed node activates all its out-neighbors in a single timestamp. The influence persistence is 1 if and only if all nodes in Y are activated.

($\Rightarrow$) Suppose $\mathcal{S}' \subset \mathcal{S}$ with $|\mathcal{S}'| = k$ covers U. If we take all nodes u_j that represent the corresponding subsets $S_i \in \mathcal{S}'$ as seed nodes, all nodes in Y can be activated.

($\Leftarrow$) Suppose $X' \subset X$ with $|X'| = k$ is the seed node set that can activate all nodes in Y. Then, the set $\mathcal{S}' = \{S_i : x_i \in X'\}$ can cover U.

Therefore, InfPM can find k seed nodes if and only if the DSC has a solution, establishing NP-hardness. □

Theorem 2. *Unless P=NP, there exists no polynomial-time constant-factor approximation algorithms for the InfPM problem.*

Proof. We prove this by reducing from the DSC problem, which is NP-complete. Given a DSC instance, we construct the InfPM instance using the same settings as in Theorem 1. Suppose there exists a polynomial-time algorithm $\mathcal{A}$ that achieves a constant-factor approximation for InfPM. That is, for any instance, $\mathcal{A}$ returns a seed node set S such that the influence persistence satisfies: $F(\alpha, \theta, S) \geq \frac{1}{c} \cdot F(\alpha, \theta, S^*)$, where $F(\alpha, \theta, S^*)$ denotes the maximal influence persistence. Then, applying $\mathcal{A}$ to the constructed instance leads to two cases: (1) If the DSC instance is a YES-instance (i.e., $\exists \mathcal{S}' \subset \mathcal{S}$ with $|\mathcal{S}'| = k$ covers U), then $F(\alpha, \theta, S^*) = 1$, and $\mathcal{A}$ outputs a seed node set S such that $F(\alpha, \theta, S) \geq \frac{1}{c} > 0$; (2) If the DSC instance is a NO-instance (i.e., no k subsets cover U), then $F(\alpha, \theta, S^*) = 0$, and $\mathcal{A}$ returns $F(\alpha, \theta, S) = 0$. Therefore, $\mathcal{A}$ can distinguish YES- and NO-instances of the DSC problem by simply verifying whether $F(\alpha, \theta, S) > 0$, allowing us decide whether the original set cover instance admits a size-k cover. This would solve DSC exactly, thereby solving an NP-complete problem in polynomial time, which contradicts the widely believed assumption that P ≠ NP. Hence, no polynomial-time constant-factor approximation algorithm exists for InfPM unless P = NP. □

Theorem 3. *Given a temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$ and parameters α and θ, the influence persistence function $F(\alpha, \theta, S)$ is monotone and non-submodular w.r.t. S under both IC and LT models.*

Proof. **(1) Monotonicity:** By the monotonicity property of influence spread [22], adding seed nodes can not decrease the expected influence spread in any snapshot under both IC and LT models. Thus, for any two seed sets $S_1 \subseteq S_2 \subseteq V$, we have $\sigma_t(S_1) \leq \sigma_t(S_2)$ for all $t \in [1, T]$. This implies $\mathcal{T}(S_1, \mathcal{G}) \subseteq \mathcal{T}(S_2, \mathcal{G})$. Consequently, any snapshot that contributes to $F(\alpha, \theta, S_1)$ will also contribute to $F(\alpha, \theta, S_2)$, ensuring that $F(\alpha, \theta, S_1) \leq F(\alpha, \theta, S_2)$. This establishes the monotonicity of $F(\alpha, \theta, S)$.

(2) Non-submodularity: For two sets $A, B \subseteq V$, if a function $f(\cdot)$ is submodular, it must satisfy $f(A)+f(B) \geq f(A \cup B)+f(A \cap B)$. Consider the temporal graph in Figure 2. Let $k = 1$, $\alpha = 50\%$, $\theta = 2$, $A = \{a\}$, and $B = \{b\}$. Then, $A \cap B = \emptyset$ and $A \cup B = \{a, b\}$. We have the influence persistence: $F(0.5, 2, \{a\}) = 2$; $F(0.5, 2, \{b\}) = 0$; $F(0.5, 2, \{a, b\}) = 3$; and $F(0.5, 2, \emptyset) = 0$. We have the following inequality: $F(0.5, 2, \{a\}) + F(0.5, 2, \{b\}) = 2 + 0 = 2 < F(0.5, 2, \{a \cup b\}) + F(0.5, 2, \emptyset) = 3 + 0 = 3$, indicating $f(A) + f(B) < f(A \cup B) + f(A \cap B)$. Therefore, $F(\alpha, \theta, S)$ is non-submodular w.r.t. S. □

4 InfPM Algorithms

This section presents ReverseGreedy and LazyReplace algorithms to address the InfPM problem.

4.1 ReverseGreedy Algorithm

Given a snapshot graph G_t, let $\sigma_t(u|S) = \sigma_t(S \cup \{u\}) - \sigma_t(S)$ denote the marginal gain of adding u to the seed set S. Traditional greedy methods for IM [22] start from an empty set $S = \emptyset$ and iteratively add nodes with the highest marginal gain. However, this strategy faces significant limitations when applied to InfPM. Most notably, a *cold-start issue* arises: when the seed set is empty or contains only a few nodes, the influence persistence is likely zero. This is because isolated nodes, or even a few nodes, often fail to generate sufficient influence to satisfy the threshold α across snapshots, preventing the formation of qualified persistent intervals. Consequently, the marginal gain for any node becomes zero, hindering effective early-stage selections. Additionally, InfPM requires evaluating marginal gains across multiple snapshots - not just one - which adds complexity. Moreover, the InfPM objective is non-submodular as demonstrated in Theorem 3, undermining the effectiveness and $(1 - 1/e)$ approximation of greedy algorithms.

To overcome these limitations, we propose ReverseGreedy (RevG), a fundamentally different strategy. Instead of building up from an empty set, RevG starts with a candidate set C (where $|C| \geq k$) that already achieves substantial influence spread. It then iteratively removes the node contributing least to the overall persistence, continuing until exactly k nodes remain. The final set is returned as

the solution. However, applying RevG effectively requires tackling two main challenges. (1) *Candidate Set Construction*: A larger candidate set offers greater potential to include high-quality seed nodes but requires more pruning iterations. A smaller set reduces computational cost but risks missing influential nodes. Therefore, constructing a candidate set that balances coverage and size is critical - it should be large enough to include good candidates, yet small enough to maintain efficiency. (2) *Snapshot Evaluation Cost*: Temporal graphs typically comprise a large number of snapshots. In each iteration, deciding which node to remove requires evaluating whether the influence spread after removal still meets the threshold α across *all snapshots*. The repeated evaluation introduces substantial computational overhead.

Constructing the initial candidate set. To address the first challenge, we design the Candidate Generation algorithm, which intelligently constructs a candidate set C^*, consisting of nodes likely to be selected as seeds. The key intuition is that nodes appearing in optimal k-seed sets that maximize influence in individual snapshots are more likely to be part of a globally effective seed set. However, directly taking the union of these seed sets may yield a large candidate set. To control its size, we introduce two pruning strategies based on the following theorems.

Theorem 4. *Given a temporal graph $\mathcal{G}$ and influence threshold α, if there exists a snapshot $G_t = (V, E_t) \in \mathcal{G}$, s.t. $\max_{S' \subset V, |S'|=k} \sigma_t(S') < \alpha \cdot |V|$, then no seed set of size k can satisfy the influence threshold in G_t, and G_t can be safely pruned.*[1]

Theorem 4 indicates that if the upper bound of influence spread in a snapshot G_t (i.e., $\max_{S' \subset V, |S'|=k} \sigma_t(S')$) is less than the threshold $\alpha \cdot |V|$, then G_t can be safely pruned. Based on Theorem 4, we define the upper-bound set of α-valid timestamps as $T_\alpha^U = \{t \in [T] \mid \max_{S \subset V, |S|=k} \sigma_t(S) \geq \alpha \cdot |V|\}$.

Theorem 5. *Given a temporal graph $\mathcal{G}$, parameters α and θ, and the upper-bound set of α-valid timestamps T_α^U, let $T_\theta^U = \cup\{I \subseteq T_\alpha^U \mid I \text{ is a maximal persistent interval with } |I| \geq \theta\}$. Then, for any timestamp $t \in T_\alpha^U \setminus T_\theta^U$, no seed set of size k can satisfy the persistence requirement at G_t, and G_t can be safely pruned.*

Theorem 5 states that if timestamp t is not contained in any maximal persistent interval, even under the upper-bound influence spread of G_t, then G_t can be safely pruned, as it cannot belong to any such interval. Notably, all snapshots pruned by Theorems 4 and 5 are guaranteed to have no contribution to influence persistence and are therefore excluded from influence spread evaluations for all candidate seed sets.

Based on these two theorems, Algorithm 1 presents the pseudo-code of the Candidate Generation (CandGen) algorithm. It initializes empty sets C^*, T_α^U, and T_θ^U (Line 1), then iterates over each snapshot G_t to compute the optimal k-seed set S_t using the greedy algorithm [22] with a $(1 - 1/e)$-approximate (Line 3). If the upper-bound influence spread $\sigma_t(S_t)$ of the snapshot G_t satisfies the influence threshold, timestamp t is added to T_α^U (Line 4). Once the number of timestamps in T_α^U exceeds θ, CandGen invokes GetPersistence to obtain T_θ^U (Line 6) and constructs C^* as the union of all S_t where $t \in T_\theta^U$ (Line 7). Otherwise, it returns an empty set (Line 8). If the resulting candidate set C^* contains more than $\alpha \cdot |V|$ nodes, its influence automatically meets threshold α, making the contribution identical regardless of which node is removed from C^*. Thus, only the top $\alpha \cdot |V|$ most frequent nodes across all S_t are retained (Lines 9-10). Finally, we set the valid timestamp set T^* as T_θ^U (Line 11), and return C^* and T^* (Line 12).

[1] Due to the limited space, all the omitted proofs can be found in our appendix [10].

Algorithm 1 Candidate Generation (CandGen)

Input: Temporal graph $\mathcal{G} = \{G_t\}_{t=1}^T$, budget k, parameters α and θ
Output: Candidate set C^*, valid timestamps set T^*

1: Initialize $C^* \leftarrow \emptyset$; $T_\alpha^U \leftarrow \emptyset$, $T_\theta^U \leftarrow \emptyset$;
2: **for** each timestamp $t = 1$ to T **do**
3: Select the optimal k-seed node set S_t from G_t;
4: **if** $\sigma_t(S_t) \geq \alpha \cdot |V|$ **then** $T_\alpha^U \leftarrow T_\alpha^U \cup \{t\}$; //Theorem 4
5: **if** $|T_\alpha^U| \geq \theta$ **then**
6: $T_\theta^U \leftarrow$ GetPersistence $(T_\alpha^U, \theta, T_\theta^U)$;
7: **for** each timestamp $t \in T_\theta^U$ **do** $C^* \leftarrow C^* \cup S_t$; //Theorem 5
8: **else** $C^* \leftarrow \emptyset$;
9: **if** $|C^*| > \alpha \cdot |V|$ **then**
10: Let C^* be the top $\alpha \cdot |V|$ most frequent nodes in $\{S_t | t \in T_\theta^U\}$;
11: $T^* \leftarrow T_\theta^U$;
12: **Return** C^*, T^*.

Procedure GetPersistence $(T_X^\alpha, \theta, T_X^\theta)$:

1: Sort all timestamp t in T_X^α in ascending order;
2: $F(\alpha, \theta, X) \leftarrow 0$, $\mathcal{T}(X, \mathcal{G}) \leftarrow \emptyset$, $s \leftarrow 0$;
3: **for** $i \leftarrow 1$ to $|T_X^\alpha|$ **do**
4: **if** $i = |T_X^\alpha|$ **or** $T_X^\alpha[i] \neq T_X^\alpha[i-1] + 1$ **then**
5: **if** $i - s \geq \theta$ **then**
6: $\mathcal{T}(X,\mathcal{G}) \leftarrow \mathcal{T}(X,\mathcal{G}) \cup \{[T_X^\alpha[s], T_X^\alpha[i-1]]\}$;
7: $T_X^\theta \leftarrow T_X^\theta \cup \{t \mid T_X^\alpha[s] \leq t \leq T_X^\alpha[i-1]\}$;
8: $s \leftarrow i$;
9: $F(\alpha, \theta, X) \leftarrow \sum_{[t_{head}, t_{tail}] \in \mathcal{T}(X,\mathcal{G})} (t_{tail} - t_{head} + 1)$;
10: **Return** $F(\alpha, \theta, X)$, T_X^θ.

The GetPersistence procedure computes the influence persistence and the corresponding θ-valid snapshot set for a given set of α-valid snapshots T_X^α. It sorts T_X^α in ascending order (Line 1), initializes variables (Line 2), then scans for consecutive intervals (Lines 3–8). Upon detecting a discontinuity (Line 4), if the current interval length is larger than θ (Line 5), it adds the interval to $\mathcal{T}(X, \mathcal{G})$ (Line 6), and inserts timestamps into T_X^θ (Line 7). The procedure then resets the interval start position (Line 8) and returns the influence persistence and updated θ-valid snapshot set (Lines 9–10).

Greedy node removal with snapshot pruning. Although the candidate set can be pruned, the temporal graph still comprises numerous snapshots. To this end, we introduce a theorem that exploits the candidate set at each iteration to identify and exclude snapshots unlikely to contribute to influence persistence in subsequent steps.

Theorem 6. *Given a snapshot $G_t \in \mathcal{G}$, a node set $A \subset V$, influence threshold α and persistence threshold θ, $\mathcal{T}(A, \mathcal{G})$ denotes the set of all maximal (α, θ, A)-persistent influence intervals of A. If timestamp $t \notin \bigcup_{I \in \mathcal{T}(A,\mathcal{G})} I$, then $\forall A' \subset A$, we have $t \notin \bigcup_{I' \in \mathcal{T}(A',\mathcal{G})} I'$.*

Theorem 6 indicates that if a timestamp t is not covered by any maximal (α, θ, C)-persistent influence intervals induced by the current candidate set C, then t will not belong to the corresponding intervals induced by subset $C' \subset C$. This means that t definitely cannot contribute to the persistence objective when any nodes are removed from C, and hence can be safely pruned from further evaluation in the node removal process.

Based on the above considerations, we propose the RevG algorithm, with its pseudo-code presented in Algorithm 2. RevG begins

Algorithm 2 ReverseGreedy (RevG)

Input: Temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, candidate set C^*, valid timestamp set T^*, budget k, parameters α and θ
Output: Seed node set S

1: Initialize $S \leftarrow \emptyset, C \leftarrow C^*, T_{\min} \leftarrow T^*$;
2: **while** $|C| > k$ **do**
3: **for** each candidate node $v \in C$ **do**
4: $C' \leftarrow C \setminus \{v\}, T^{\alpha}_{C'} \leftarrow \emptyset, T^{\theta}_{C'} \leftarrow \emptyset$;
5: **for** each timestamp $t \in T_{\min}$ **do** //Theorem 6
6: **if** $\sigma_t(C') \geq \alpha \cdot |V|$ **then** $T^{\alpha}_{C'} \leftarrow T^{\alpha}_{C'} \cup \{t\}$;
7: **if** $T^{\alpha}_{C'} \neq \emptyset$ **then** $F, T^{\theta}_{C'} \leftarrow$ GetPersistence$(T^{\alpha}_{C'}, \theta, T^{\theta}_{C'})$;
8: **else** $F(\alpha, \theta, C') \leftarrow 0$;
9: $\Delta(v|C) \leftarrow F(\alpha, \theta, C) - F(\alpha, \theta, C')$;
10: **if** $\Delta(v|C) = 0$ **then break;**
11: $v^* \leftarrow \arg\min_{v \in C} \Delta(v|C), C \leftarrow C \setminus \{v^*\}$;
12: Update $F(\alpha, \theta, C) \leftarrow F(\alpha, \theta, C \setminus \{v^*\}), T_{\min} \leftarrow T^{\theta}_{C \setminus \{v^*\}}$;
13: $S \leftarrow C$;
14: **Return** S.

by initializing $S = \emptyset$, $C = C^*$, and, $T_{\min} = T^*$, which records the snapshots that contribute to persistence (Line 1). The main loop (Lines 2–11) iteratively removes one node from C until only k nodes remain. For each node $v \in C$, it iterates over each snapshot G_t to check whether the influence spread after removing v still meets the threshold α (Lines 5–6). If $T^{\alpha}_{C'}$ is non-empty, it calls GetPersistence to compute $F(\alpha, \theta, C')$ and update $T^{\theta}_{C'}$ (Line 7); otherwise, persistence is set to zero (Line 8). It then calculates the marginal loss $\Delta(v|C)$ for each node (Line 9). Once $\Delta(v|C) = 0$, the for loop (Lines 3-9) can be terminated early. Subsequently, it removes the node v^* with the smallest marginal loss (Line 10), and updates the current persistence $F(\alpha, \theta, C)$ and timestamps set $T_{\min}$ (Line 11). This process repeats until the candidate set is reduced to exactly k nodes. Finally, the remaining nodes in C are assigned to the seed node set S, which is returned as the solution (Lines 12-13).

Example 2. *Consider the temporal graph $\mathcal{G}$ with eight snapshots in Figure 3, with $\alpha = 0.6$, $\theta = 3$, and $k = 2$. CanGen first selects the optimal 2-seed node set S_i that maximizes influence spread for each snapshot $G_i \in \mathcal{G}$. Since $\sigma(S_6) = 5 < 0.6 \cdot 10 = 6$, snapshot G_6 is pruned. It then computes persistent intervals over $T_\alpha = \{1, 2, 3, 4, 5, 7, 8\}$. Since interval $[7, 8]$ has length shorter than $\theta = 3$, snapshots G_7 and G_8 are discarded, yielding $T_\theta = \{1, 2, 3, 4, 5\}$. The union of all selected seed nodes over snapshots in T_θ forms the initial candidate set $C^* = \{v_0, v_1, v_4, v_9\}$ with influence persistence of $F(0.6, 3, C^*) = 5$. RevG then iteratively removes nodes from C^*. It first removes v_4, as this causes no reduction in persistence, with $F(0.6, 3, C^*) = 5$, making it unnecessary to examine the following node v_9. It then removes v_1 with the minimum marginal loss, resulting in the final valid interval $[3, 5]$ with $F(0.6, 3, C^*) = 3$. Thus, the final seed node set is $S = \{v_0, v_9\}$.*

Time complexity. The time complexity of constructing the candidate set C^* is $O(|V| \cdot k \cdot T \cdot \varphi)$, where φ is the cost of computing influence per snapshot (Section 5). RevG then runs for $|C^*| - k$ iterations, each evaluating the persistence loss of all nodes in C^* with cost $O(T \cdot \varphi)$, resulting in a total complexity of $O(|C^*| \cdot (|C^*| - k) \cdot T \cdot \varphi)$.

4.2 LazyReplace Algorithm

In this subsection, we introduce LazyReplace (LRep), a new algorithm that identifies the optimal seed set through a replacement-based approach to maximize influence persistence.

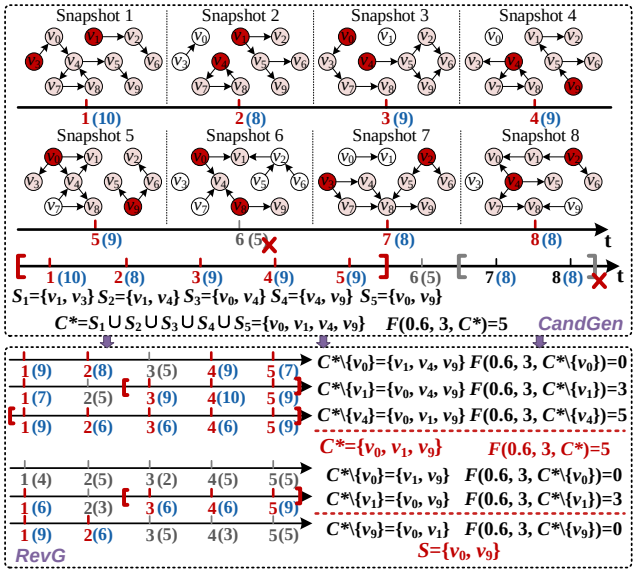


Figure 3: An example of RevG Algorithm

The core idea of LRep is to initialize a seed set with nodes that consistently exhibit high influence spread across all snapshots, then iteratively replace seed nodes with candidates that improve influence persistence. However, exhaustively evaluating all possible replacements incurs significant computational overhead. To address this, LRep adopts a lazy evaluation strategy with early termination: it prioritizes replacing the weakest node (i.e., the one contributing least to influence persistence) in the current seed set with each candidate. If this replacement fails to improve influence persistence, all subsequent replacements involving that candidate are skipped, thereby avoiding redundant calculations across all snapshots. The algorithm leverages the intuition that nodes with consistently strong influence are more likely to contribute to the optimal solution, while the lazy replacement strategy ensures efficiency by avoiding unnecessary evaluations.

LRep addresses three key issues: (1) How to efficiently construct an initial seed node set for replacement? (2) How to prioritize candidate nodes to accelerate convergence? and (3) How to identify the weakest contributor in the current seed set for lazy replacement? First, we adopt a heuristic that computes the influence spread of each node $v \in V$ across all snapshots $G_t \in \mathcal{G}$ and selects the top-k nodes with the highest average influence. This approach is based on the intuition that nodes with greater average influence are more likely to contribute significantly to overall persistence, which is empirically validated as the optimal baseline in Section 7. Second, we rank the remaining candidate nodes in descending order of their average single-node influence and perform replacements following this order, prioritizing nodes with higher potential influence spread. Third, following the principle used in RevG, we define the weakest contributor as the node whose removal results in the smallest decrease in influence persistence.

Algorithm 3 presents the pseudo-code of LRep, which identifies the optimal seed set for maximizing influence persistence through two phases. In Phase I (Lines 1–4), LRep computes each node's average influence $\bar{\sigma}(\{v\})$ across all snapshots, sorts nodes in descending order of average influence, selects the top-k nodes to form the initial

Algorithm 3 LazyReplace (LRep)

Input: Temporal graph $\mathcal{G}=\{G_t\}_{t=1}^{T}$, budget k, parameters α and θ
Output: Seed node set S

1: **for** each node $v \in V$ **do**
2: Compute average influence $\bar{\sigma}(\{v\}) \leftarrow \frac{1}{T}\sum_{t=1}^{T}\sigma_t(\{v\})$;
3: $S \leftarrow$ top-k nodes from V sorted by $\bar{\sigma}(\{v\})$ in descending order;
4: $F(\alpha,\theta,S) \leftarrow$ ComputePersistence$(\mathcal{G},S,\alpha,\theta)$;
5: **for** each candidate node $v \in V \setminus S$ **do**
6: $s^* \leftarrow$ FindWeakestNode(S);
7: $S' \leftarrow S \setminus \{s^*\} \cup \{v\}$;
8: $F'(\alpha,\theta,S') \leftarrow$ ComputePersistence$(\mathcal{G},S',\alpha,\theta)$;
9: **if** $F'(\alpha,\theta,S') \le F(\alpha,\theta,S)$ **then**
10: **Continue;** `// Early pruning: skip exhaustive search`
11: $\hat{S} \leftarrow S, \hat{F}(\alpha,\theta,\hat{S}) \leftarrow F(\alpha,\theta,S)$;
12: **for** each seed node $s \in S$ **do**
13: $\tilde{S} \leftarrow S \setminus \{s\} \cup \{v\}$;
14: $\tilde{F}(\alpha,\theta,\tilde{S}) \leftarrow$ ComputePersistence$(\mathcal{G},\tilde{S},\alpha,\theta)$;
15: **if** $\tilde{F}(\alpha,\theta,\tilde{S}) > \hat{F}(\alpha,\theta,\hat{S})$ **then**
16: $\hat{S} \leftarrow \tilde{S}, \hat{F}(\alpha,\theta,\hat{S}) \leftarrow \tilde{F}(\alpha,\theta,\tilde{S})$;
17: Update $S \leftarrow \hat{S}, F(\alpha,\theta,S) \leftarrow \hat{F}(\alpha,\theta,\hat{S})$;
18: **Return** S.
Function ComputePersistence$(\mathcal{G},S,\alpha,\theta)$:
19: $T_S^{\alpha} \leftarrow \emptyset, T_S^{\theta} \leftarrow \emptyset$;
20: **for** each timestamp $t = 1$ to T **do**
21: **if** $\sigma_t(S) \ge \alpha \cdot |V|$ **then** $T_S^{\alpha} \leftarrow T_S^{\alpha} \cup \{t\}$;
22: **if** $T_S^{\alpha} \neq \emptyset$ **then Return** GetPersistence$(T_S^{\alpha},\theta,T_S^{\theta})$;
23: **else Return** 0.

seed set S, and calculates the initial influence persistence $F(\alpha,\theta,S)$. Phase II (Lines 5–18) iteratively refines S through lazy evaluation with early pruning. For each candidate node $v \in V \setminus S$, LRep first identifies the weakest contributor s^* in the current seed set by invoking Lines 3–9 of Algorithm 2 (Line 6), then tests whether replacing s^* with v improves persistence. If no improvement is detected (Line 9), all further exhaustive evaluations involving v are pruned (Line 10). Otherwise, it exhaustively evaluates all replacements within S, tracking the best seed set $\hat{S}$ and corresponding persistence $\hat{F}(\alpha,\theta,\hat{S})$ (Lines 12–16), and finally updates S and $F(\alpha,\theta,S)$ (Line 17). The auxiliary function ComputePersistence (Lines 19–23) calculates the influence persistence of a given seed node set S.

Example 3. *Consider the temporal graph $\mathcal{G}$ with five snapshots in Figure 4, with $\alpha = 0.6$, $\theta = 3$, and $k = 2$. LRep first computes each node's average influence across snapshots and selects the initial seed set $S = \{v_0, v_2\}$ as the top-2 influential nodes with $\bar{\sigma}(\{v_0\}) = 2.4$ and $\bar{\sigma}(\{v_2\}) = 2.2$. Then, for each candidate in $C = \{v_1, v_3, v_4\}$ (sorted by descending average influence), LRep replaces each seed node in S with the candidate. For example, replacing v_2 with v_1 improves influence persistence from 0 to 3. Finally, when $\tilde{S} = \{v_3, v_1\}$ the influence persistence equals 5 and LRep returns $S = \{v_3, v_1\}$.*

Time complexity. Estimating average influence over T snapshots requires $O(|V| \cdot T \cdot \varphi)$ time. For each of the $|V| - k$ candidates, LRep performs up to $k + 1$ evaluations, each costing $O(T + \varphi \cdot k)$. Thus, the overall complexity of LRep is $O(|V| \cdot T \cdot \varphi + |V| \cdot k(T + \varphi \cdot k))$.

5 Influence Computation

Both RevG and LRep require numerous computations of influence spread across multiple snapshots to evaluate whether a given node set satisfies the influence threshold. However, computing the exact influence spread $\sigma_t(S)$ for any node set $S \subset V$ and snapshot $G_t \in \mathcal{G}$ is #**P**-hard under both the IC and LT models, since each snapshot is essentially a static graph and this complexity has been established

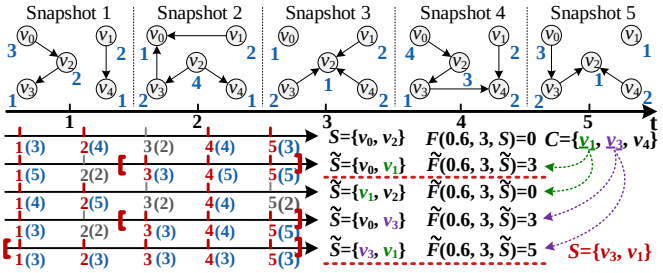


Figure 4: An example of LRep Algorithm

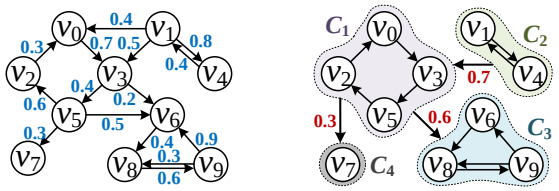
in prior works [11, 22]. Therefore, we employ approximation methods to estimate the value of $\sigma_t(S)$. In the following, we focus on estimating influence spread on a single snapshot, noting that this approach can be applied to the entire temporal graph.

Recent studies have employed sampling-based approaches for influence spread estimation, ranging from Monte-Carlo (MC) simulations [22] to Reverse Influence Sampling (RIS) [5]. Given that RIS significantly improves the efficiency of estimating influence spread compared to MC simulations, we adopt RIS to estimate $\sigma_t(S)$, consistent with state-of-the-art methods in influence estimation [17, 40]. Each reverse reachable (RR) set R sampled by RIS represents a subset of V generated through the following process:

(1) Generate a random graph G_t' from G_t by retaining all nodes while independently removing each edge $e_t = (u, v)$ with probability $1 - w_t(u, v)$;
(2) Select a node v uniformly at random from G_t';
(3) Let R be the set of all nodes u from which there exists a directed path to v in G_t'.

Note that the above describes RR set generation under the IC model. Due to space constraints, the generation process under the LT model can be found in [40]. Given a snapshot G_t, a seed set S, and a random RR set R generated on G_t, we define a random variable $\Upsilon_t(S,R)$ such that $\Upsilon_t(S,R) = 1$ if $S \cap R \neq \emptyset$ and $\Upsilon_t(S,R) = 0$ otherwise. Borgs et al. [5] proved that $\sigma_t(S)$ equals $\mathbb{E}[\Upsilon_t(S,R)] \cdot |V|$ under the triggering model, which generalizes both the IC and LT models. Given a set $\mathcal{R}_t = \{R_1, R_2, \ldots\}$ of RR sets sampled from snapshot G_t, $\mathbb{E}[\Upsilon_t(S,R)] \cdot |V|$ can be unbiasedly estimated using the empirical mean $\frac{\sum_{R\in\mathcal{R}_t}\Upsilon_t(S,R)}{|\mathcal{R}_t|} \cdot |V|$ based on concentration bounds. Therefore, when $\mathcal{R}_t$ is fixed, we define $f_{\mathcal{R}_t}(S) = \frac{\sum_{R\in\mathcal{R}_t}\Upsilon_t(S,R)}{|\mathcal{R}_t|} \cdot |V|$ as an unbiased estimator of $\sigma_t(S)$ under the RIS framework.

While RIS enables efficient influence estimation for a given seed set S on a single snapshot, it faces significant challenges when applied to temporal graphs. In temporal graphs, each snapshot exhibits distinct structural characteristics that necessitate independent influence estimations for the seed set S at every timestamp. This computational cost becomes prohibitive in real-world scenarios where rapid network dynamics generates *extensive sequences of snapshots* and the graphs are inherently *large*. To this end, we propose a novel snapshot compression technique that significantly reduces the size of each snapshot, thereby shortening sampling paths and minimizing the number of nodes visited per RR set under RIS. Our approach first performs edge sampling and then applies strongly connected component (SCC) decomposition to simplify the graph structure by aggregating nodes into SCCs. Each snapshot G_t is thereby transformed into a compressed directed acyclic graph, referred to as SCC-DAG and denoted by D_t. Based on this structure,



(a) Snapshot graph G_t (b) SCC-DAG D_t

Figure 5: Illustration of the snapshot compression method

we extend the classical RIS method to operate on SCC-DAGs by sampling SCC-based RR sets, referred to as SCC-RR sets. We further develop an unbiased influence estimator based on SCC-RR sets, which provides theoretical guarantees. This framework involves addressing three key challenges: (1) designing an effective node partition strategy to transform snapshots into compact SCC-DAGs; (2) developing efficient algorithms for sampling random SCC-RR sets from the compressed structure; and (3) establishing an unbiased estimator for $\sigma_t(S)$ with theoretical guarantees.

Constructing compressed snapshots SCC-DAGs. We begin by introducing a method for merging nodes into SCCs. Since computing strong connectivity reliability is #**P**-hard [3, 44], and even approximate estimation via sampling is computationally expensive, we adopt a heuristic approach. Given a snapshot $G_t \in \mathcal{G}$, we sample r random graphs by retaining all nodes while independently removing each edge $e_t = (u, v)$ with probability $1 - w_t(u, v)$. We then apply SCC decomposition to each sampled graph and select the partition with the fewest components (i.e., the coarsest). The intuition is that fewer SCCs imply larger strongly connected regions, better preserving influence propagation potential. Among the sampled graphs, this partition approximated an upper-bound structure of strong connectivity. We formalize this in Definition 3. Given the SCC partition $P_t \in \mathcal{P}$ for each snapshot G_t, we construct its corresponding SCC-DAG D_t [36], as stated in Definition 4.

Definition 3. **(SCC partition P_t).** *Given a snapshot $G_t \in \mathcal{G}$, let $G_t^1 = (V, E_t^1), \ldots, G_t^r = (V, E_t^r)$ be r random graphs independently sampled from G_t, and let $P_t^i = \{C_1, \ldots, C_\ell\}$ denote the SCC decomposition of G_t^i. Then, the SCC partition of the node set V is chosen such that $|P_t^{i^*}| = \min_{i \in [r]} |P_t^i|$.*

Definition 4. **(SCC-DAG).** *Given a snapshot G_t and its corresponding SCC partition $P_t = \{C_j\}_{j \in [\ell]}$, the compressed snapshot (SCC-DAG) derived from G_t is defined as $D_t(\Lambda_t, \Phi_t, p_t)$, where:*

$$\Lambda_t = \{C_j \mid j \in [\ell]\},$$
$$\Phi_t = \{\tilde{e}_t = (C_i, C_j) \mid C_i \neq C_j,\ \exists e_t = (u, v) \in E_t,\ u \in C_i,\ v \in C_j\},$$
$$p_t(C_i, C_j) = 1 - \prod_{\substack{e_t=(u,v)\in E_t,\\ u\in C_i,\ v\in C_j}} (1 - w_t(u, v)) \quad \forall \tilde{e}_t = (C_i, C_j) \in \Phi_t.$$

Here, Λ_t represents the node set of hypernodes (SCCs), Φ_t denotes the edge set between distinct components, and $p_t(C_i, C_j)$ is the aggregated influence weight from component C_i to C_j.

Example 4. *Figure 5 illustrates a snapshot graph $G_t(V, E_t)$ and its corresponding SCC-DAG $D_t(\Lambda_t, \Phi_t, p_t)$. In this example, assuming the SCC partition is $P_t = \{C_1, C_2, C_3, C_4\}$, then we have $\Lambda_t = \{C_1, C_2, C_3, C_4\}$ where $C_1 = \{v_0, v_2, v_3, v_5\}$, $C_2 = \{v_1, v_3\}$, $C_3 = \{v_6, v_8, v_9\}$, and $C_4 = \{v_7\}$. The inter-component edges are represented as $\Phi_t = \{(C_1, C_4), (C_1, C_3), (C_2, C_1)\}$. The influence weight on an edge (C_i, C_j) in D_t aggregates the influence weights of all edges connecting nodes from components C_i to C_j in the original graph G_t. For instance, there are two edges connecting C_1 to C_3 with individual influence weights of 0.2 and 0.5, respectively. The aggregated influence weight of edge (C_1, C_3) is calculated as $1 - (1 - 0.5)(1 - 0.2) = 0.6$.*

Algorithm 4 SCC-DAG based influence estimation framework

Input: A snapshot graph $G_t \in \mathcal{G}$, a seed node set S, and a sample size $\mathcal{I}_t$
Output: Estimated influence spread $f_{\tilde{\mathcal{R}}_t}(S)$

1: $P_t \leftarrow$ Compute SCC partition of G_t by Definition 3;
2: $D_t \leftarrow$ Build SCC-DAG $D_t(\Lambda_t, \Phi_t, p_t)$ from G_t and P_t by Definition 4;
3: $\tilde{\mathcal{R}}_t \leftarrow$ Sample $\mathcal{I}_t$ random SCC-RR sets from D_t;
4: $f_{\tilde{\mathcal{R}}_t}(S) \leftarrow$ Estimate influence spread of S using $\tilde{\mathcal{R}}_t$ by Equations 4 and 5;
5: **Return** $f_{\tilde{\mathcal{R}}_t}(S)$.

Sampling random SCC-RR sets on SCC-DAG. Building upon the above definitions, we extend the classical RIS framework to sample SCC-based reverse reachable (SCC-RR) sets on compressed SCC-DAG. Given an SCC-DAG $D_t = (\Lambda_t, \Phi_t, p_t)$, each sampled SCC-RR set $\tilde{R} \subseteq \Lambda_t$ is generated as follows:

(1) Generate a random graph D_t' from D_t by retaining all hypernodes (i.e., SCCs) while independently removing each hyperedge $\tilde{e}_t = (C_i, C_j)$ with probability $1 - p_t(C_i, C_j)$;
(2) Select a hypernode C_i from D_t' with *probability* $|C_i|/|V|$;
(3) Let $\tilde{R}$ be the set of all hypernodes C_j such that there exists a directed path to C_i in D_t'.

Since the SCCs $\{C_1, C_2, \ldots, C_\ell\}$ form a partition of the node set V, selecting a hypernode C_i with probability $|C_i|/|V|$ is mathematically equivalent to uniform node sampling in the original graph G_t. This enables us to build SCC-RR sets on the SCC-DAG while preserving the uniform sampling assumption of the traditional RIS, ensuring that the influence estimator remains unbiased.

Estimating $\sigma_t(S)$ via sampled SCC-RR sets. Given a set of SCC-RR set $\tilde{\mathcal{R}}_t = \{\tilde{R}_1, \tilde{R}_2, \ldots\}$ sampled from D_t, we present an unbiased method for estimating the influence spread of a seed node set S. For a snapshot graph G_t, a seed node set S, and a random SCC-RR set $\tilde{R}$, we define a random variable $\Omega_t(S, \tilde{R})$ as follows:

$$\Omega_t(S, \tilde{R}) = \begin{cases} 1 & \text{if } \exists s \in S,\ \exists C_i \in \tilde{R},\ \text{s.t. } s \in C_i, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

This variable captures whether the seed set S intersects with the SCC-RR set $\tilde{R}$. The formulation is valid because strong connectivity within each hypernode $C_i \in \tilde{R}$ ensures that if any node in C_i is influenced by S, then all nodes in C_i are also influenced. For the entire collection $\tilde{\mathcal{R}}_t$, we define $\Omega_t(S, \tilde{\mathcal{R}}_t) = \sum_{\tilde{R} \in \tilde{\mathcal{R}}_t} \Omega_t(S, \tilde{R})$. Based on this, we construct an unbiased estimator $f_{\tilde{\mathcal{R}}_t}(S)$ for $\sigma_t(S)$:

$$f_{\tilde{\mathcal{R}}_t}(S) = \frac{\Omega_t(S, \tilde{\mathcal{R}}_t)}{|\tilde{\mathcal{R}}_t|} \cdot |V| \tag{5}$$

We now establish that the estimate $f_{\tilde{\mathcal{R}}_t}(S)$, obtained through SCC-based sampling, serves as a concentration bound with a high probability, as demonstrated in Theorem 7.

Theorem 7. *For any seed set $S \subset V$ and user-defined parameters $\epsilon, \delta \in (0, 1)$, with probability at least $(1 - \delta)$, if $|\tilde{\mathcal{R}}_t| \geq \frac{(2+\epsilon)\cdot \log \delta}{\epsilon^2 \cdot \tilde{\sigma}_t(S)}$, then the sampling-based estimate $f_{\tilde{\mathcal{R}}_t}(S)$ satisfies*

$$\left| f_{\tilde{\mathcal{R}}_t}(S) - \tilde{\sigma}_t(S) \right| < \epsilon \cdot \tilde{\sigma}_t(S) \tag{6}$$

By the law of large numbers, as the number of samples increases, the estimation error of the expected influence decreases and stays within the ϵ-bound.

Algorithm 4 presents our complete SCC-DAG based influence estimation framework. The framework first constructs the SCC partition and corresponding SCC-DAG for each snapshot, then performs sampling and estimation on the compressed graph structure. Since the SCC-DAG D_t is typically much smaller than the original snapshot G_t, our framework substantially improves computational efficiency compared to directly applying RIS on the original snapshot. The theoretical guarantee for our framework is established by bounding the relative error of the estimation as follows.

Theoretical analysis. Consider a snapshot $G_t(V, E_t)$ with a node partition $P_t = \{C_j\}_{j\in[\ell]}$ and its corresponding SCC-DAG $D_t(\Lambda_t, \Phi_t, p_t)$. We set the influence weights as follows: $\tilde{w}_t^{in}(u, v) = 1$ for any two nodes $u, v \in C_j$ within the same strongly connected component $C_j \in \Lambda_t$, and $\tilde{w}_t^{out}(u, v) = p_t(C_i, C_j)$ for nodes in different components where $u \in C_i$ and $v \in C_j$ with $i \neq j$. Given a seed set S, let $\tilde{\sigma}_t(S)$ and $\sigma_t(S)$ denote the influence spread on the compressed graph D_t and original graph G_t, respectively. Based on the above construction, we establish a lower bound for $\tilde{\sigma}_t(S)$ w.r.t. $\sigma_t(S)$.

Lemma 1. *For any $S \subset V$, $\sigma_t(S) \leq \tilde{\sigma}_t(S)$.*

Then, we define the *strongly connected reliability* of each SCC-induced subgraph $G_t[C_j] = (V_j, E_j)$ as the probability that a randomly sampled subgraph of $G_t[C_j]$ is strongly connected. Formally,

$$\mathrm{Scr}(G_t[C_j]) = \sum_{X_j \subseteq E_j} \left(\prod_{e\in X_j} w_t(e) \cdot \prod_{e\in E_j \setminus X_j} (1 - w_t(e)) \cdot \kappa \right). \quad (7)$$

$\kappa = 1$ if the sampled subgraph (V_j, X_j) is strongly connected and $\kappa = 0$ otherwise. We now present an upper bound for $\tilde{\sigma}_t(S)$ using the strongly connected reliabilities of all components.

Lemma 2. *For any $S \subset V$, $\tilde{\sigma}_t(S) \leq \left(\prod_{j=1}^{\ell} \mathrm{Scr}(G_t[C_j])^{-1}\right) \cdot \sigma_t(S)$.*

By Theorem 7 and Lemmas 1 and 2, we have Theorem 8.

Theorem 8. *Given a snapshot graph G_t and a seed node set $S \subset V$, let $f_{\tilde{\mathcal{R}}_t}(S)$ denote the estimator of $\tilde{\sigma}_t(S)$. Then the relative error between $f_{\tilde{\mathcal{R}}_t}(S)$ and $\sigma_t(S)$ is bounded by:*

$$(1 - \epsilon) \cdot \sigma_t(S) \leq f_{\tilde{\mathcal{R}}_t}(S) \leq \left(\frac{1 + \epsilon}{\prod_{j=1}^{\ell} \mathrm{Scr}(G_t[C_j])} \right) \cdot \sigma_t(S). \quad (8)$$

Time complexity. The time complexity analysis consists of three main components: (1) SCC partition requires $O(r \cdot (|V| + |E_t|))$ time for r sampled random graphs; (2) SCC-DAG construction takes $O(|E_t|)$ time; (3) sampling $\tilde{\mathcal{R}}_t$ costs $O(\mathcal{I}_t \cdot \frac{|\Phi_t|}{|\Lambda_t|} \cdot \sigma_t(C'))$, where C' is a random component selected by in-degree from D_t [43]. The overall expected time per snapshot is $O(\mathcal{I}_t \cdot \frac{|\Phi_t|}{|\Lambda_t|} \cdot \sigma_t(C') + r \cdot (|V| + |E_t|))$, yielding a total complexity of $O(\sum_{t=1}^{T} (\mathcal{I}_t \cdot \frac{|\Phi_t|}{|\Lambda_t|} \cdot \sigma_t(C') + r \cdot (|V| + |E_t|))$ across all snapshots.

6 Win-InfPM Problem

In this section, we introduce a variant of the InfPM problem and present an effective method to address it.

In certain real-world scenarios such as repeated product recommendations or public information campaigns, influence can remain effective even with intermittent exposure, as long as the influence threshold is reached at multiple timestamps within a given period. To capture such cases, we introduce a relaxed temporal influence model, which requires influence to exceed α at θ distinct timestamps within a larger window W (with $W > \theta$). To this end, we define the concept of maximal (α, θ, W, S)-persistent influence interval.

Algorithm 5 Prefix-Sum Sliding-Window (PssW)

Input: Temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, a seed node set $\hat{S}$, window size W, α-valid timestamps set $t \in T_{\hat{S}}^{\alpha}$, parameters α and θ,
Output: Window-based influence persistence $F(\alpha, \theta, W, \hat{S})$

1: Initialize prefixSum[0 . . . T] ← 0;
2: **for** each timestamp t = 1 to T **do** //Build prefix sum array
3: valid[t] ← $\mathbf{1}[t \in T_{\hat{S}}^{\alpha}]$;
4: prefixSum[t] ← prefixSum[$t-1$] + valid[t];
5: $F(\alpha, \theta, W, \hat{S}) \leftarrow 0$, $\mathcal{T}_W(\hat{S}, \mathcal{G}) \leftarrow \emptyset$, $ts \leftarrow -1$, $te \leftarrow -1$;
6: **for** i = 1 to $T - W + 1$ **do** //Detect persistent intervals
7: cnt ← prefixSum[$i + W - 1$] − prefixSum[$i - 1$];
8: **if** cnt ≥ θ **then**
9: **if** $ts = -1 \wedge te = -1$ **then** $ts \leftarrow i$, $te \leftarrow i + W - 1$;
10: **else if** $te + 1 = i + W - 1$ **then**
11: $te \leftarrow i + W - 1$; //Extend current interval
12: **else if** $ts \neq -1 \wedge te \neq -1$ **then**
13: $\mathcal{T}_W(\hat{S}, \mathcal{G}) \leftarrow \mathcal{T}_W(\hat{S}, \mathcal{G}) \cup \{[ts, te]\}$, $ts \leftarrow -1$, $te \leftarrow -1$;
14: **if** $ts \neq -1 \wedge te \neq -1$ **then** $\mathcal{T}_W(\hat{S}, \mathcal{G}) \leftarrow \mathcal{T}_W(\hat{S}, \mathcal{G}) \cup \{[ts, te]\}$;
15: $F(\alpha, \theta, W, \hat{S}) \leftarrow \sum_{(t_h, t_t) \in \mathcal{T}_W(\hat{S}, \mathcal{G})} (t_t - t_h + 1)$;
16: **Return** $F(\alpha, \theta, W, \hat{S})$.

Definition 5. **(Maximal (α, θ, W, S)-persistent influence interval).** *Given a temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, a seed node set $S \subset V$, a time window $W > \theta$, and two parameters $\alpha \in (0, 1]$ and $\theta \geq 1$ ($\theta \in \mathbb{N}^+$), a time interval $[t_s, t_e] \subseteq [1, T]$ is a maximal (α, θ, S, W)-window persistent influence interval for $\mathcal{G}$ if and only if:*

1) ***(α, θ, W)-persistence:*** *$\forall t \in [t_s, t_e], |\{t' \in [t, \min\{t+W-1, T\}] \mid \sigma_{t'}(S) \geq \alpha \cdot |V|\}| \geq \theta$.*
2) ***Maximalist:*** *$\nexists\, [t'_s, t'_e] \supset [t_s, t_e]$ satisfies above condition.*

Based on Definitions 2 and 5, we define $F(\alpha, \theta, W, S)$ as the window-based influence persistence of S in temporal graph $\mathcal{G}$, calculated as the total length of all maximal (α, θ, S, W)-window persistent influence intervals. Then, we formally define the Window-based InfPM (Win-InfPM) problem below.

Problem 2. **(Win-InfPM).** *Given a temporal graph $\mathcal{G} = \{G_t\}_{t=1}^{T}$, a budget k, two parameters $\alpha \in (0, 1]$ and $\theta \geq 1$, and a time window $W > \theta$, Win-InfPM aims to find a seed node set $\hat{S} \subset V$ with $|\hat{S}| = k$ such that the window-based influence persistence is maximized. Formally:*

$$\hat{S} = \arg\max_{\hat{S}\subset V \wedge |\hat{S}|=k} F(\alpha, \theta, W, \hat{S}). \quad (9)$$

To address the Win-InfPM problem, we introduce a new evaluation procedure, Prefix-Sum Sliding-Window (PssW), into the proposed RevG and LRep frameworks. PssW efficiently computes the window-based influence persistence $F(\alpha, \theta, W, \hat{S})$ for a given seed set $\hat{S}$. The pseudo-code is shown in Algorithm 5. First, PssW builds a prefix sum array by checking each timestamp t for α-valid, i.e., $t \in T_{\hat{S}}^{\alpha}$, meaning the influence spread of $\hat{S}$ in G_t exceeds α, and cumulatively counts these valid timestamps (Lines 1–4). Second, it slides a window of length W over time, using prefix sum subtraction to efficiently compute the number of α-valid timestamps in each window (Line 7). A window is marked as persistent if it contains at least θ α-valid timestamps, and each persistent window is extended to include adjacent windows that also satisfy the condition, forming maximal intervals stored in $\mathcal{T}_W(\hat{S}, \mathcal{G})$ (Lines 5–14). Finally, it returns the total influence persistence (Lines 15–16).

Time Complexity. Constructing the prefix sum array over T snapshots, scanning the $T - W + 1$ sliding windows, and aggregating valid interval lengths each take $O(T)$ time in the worst case. Thus, the overall time complexity of Algorithm 5 is $O(T)$.

Table 1: Dataset Statistics

Dataset	$\|V\|$	$\|E\|$	T	Time scale
CollegeMsg (CM)	1,899	59,835	193	Day
EmailCore (EC)	986	332,334	803	Day
MathOverflow (MO)	24,818	506,550	2,350	Day
AskUbuntu (AU)	159,316	964,437	2,613	Day
SuperUser (SU)	194,085	1,443,339	2,773	Day
WikiTalk (WT)	1,140,149	7,833,140	696	Day
StackOverflow (SO)	2,601,977	63,497,050	277	Day

Table 2: Exact VS. Approximate on EC-Dept3

Parameter	$k=3$			$k=4$		
Methods	**Enum**	**RevG**	**LRep**	**Enum**	**RevG**	**LRep**
Influence persistence	31	24	28	74	63	71
Running time (sec)	444.06	0.6014	0.6308	13846	0.6186	0.6417
Approximation ratio	/	77.4%	90.3%	/	85.1%	95.9%
Parameter	$k=5$			$k=6$		
Methods	**Enum**	**RevG**	**LRep**	**Enum**	**RevG**	**LRep**
Influence persistence	142	125	128	/	198	219
Running time (sec)	305722	0.6235	0.6674	> 7 days	0.6241	0.6901
Approximation ratio	/	88.0%	90.1%	/	/	/

7 Experiments

In this section, we empirically evaluate our proposed algorithms. All experiments are implemented in C++ and performed on a Linux machine with Intel(R) Core(TM) 3.70GHz CPU and 128GB of RAM.

7.1 Experimental Settings

Datasets. We conduct experiments on seven real-world temporal social networks, all publicly available from SNAP [28]. Dataset statistics are summarized in Table 1. Specifically, CM is an online messaging network, and EC is an email communication network. MO, AU, SU, and SO are temporal user interaction networks from various Q&A platforms. WT is a temporal network where nodes are Wikipedia users and edges indicate edits to each other's Talk pages. For each network, we adopt the widely used *weighted-cascade* model [22] to assign influence weights $w_t(u,v)$ on edges in each snapshot G_t, where $w_t(u,v) = 1/|N_t^{in}(v)|$.

Algorithms. We compare a set of algorithms in experiments.

- **RevG** and **LRep**: Our ReverseGreedy and LazyReplace algorithms that utilize the reverse influence sampling (RIS) method [5], a state-of-the-art technique for estimating influence spread.
- **RevG+** and **LRep+**: Our ReverseGreedy and LazyReplace algorithms that employ the influence spread computation method proposed in Section 5.
- **Influential (INF)**: A method that selects seed nodes with the highest average expected influence across all snapshots.
- **HighDegree (DEG)**: A method that selects seed nodes with the highest average out-degrees across all snapshots.
- **Random (RAN)**: A method that uniformly selects seed nodes.
- **Enumerate (Enum)**: An exact method that enumerate all size-k seed node sets, computes their influence persistence, and returns the one with the maximum influence persistence.

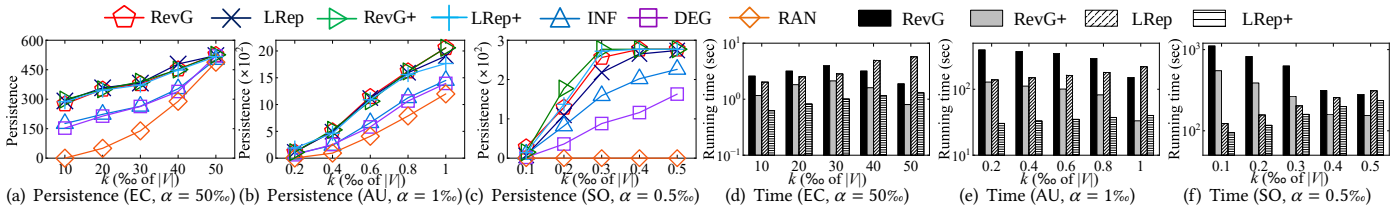
Parameter and Metric Settings. We evaluate the performance of algorithms across different parameters, including the budget k, influence threshold α, persistence threshold θ, the number of samples per snapshot $\mathcal{I}_t$, and time span T. In each experiment, only one parameter is varied while the others are fixed at their default values: $\theta = 2$ for all datasets; $\mathcal{I}_t$ is set to 6×10^3 for the RIS-based approach and 3×10^3 for our SCC-DAG-based method; T is set to the entire time scale of the temporal graph; k and α are set differently for different datasets. Notably, the maximum value of α is set as the ratio of the average number of active users across all snapshots to the total number of users, where active users are those involved in at least one interaction (i.e., non-isolated) in a snapshot. We ensure that $k < \alpha \cdot |V|$ to prevent all snapshots from automatically satisfying the α-condition. We report the influence persistence and running time (second) for each experiment. As with [45, 48], we employ Monte Carlo simulation to compute the expected influence spread for the returned seed node set. For each experiment, we repeat each algorithm five times and report the average results.

7.2 Evaluation of InfPM Algorithm

This subsection evaluates the InfPM algorithms. Due to space constraints and similar results across datasets, we report results for EC, AU, and SO in this paper. Results for all datasets are available in our extended version [10].

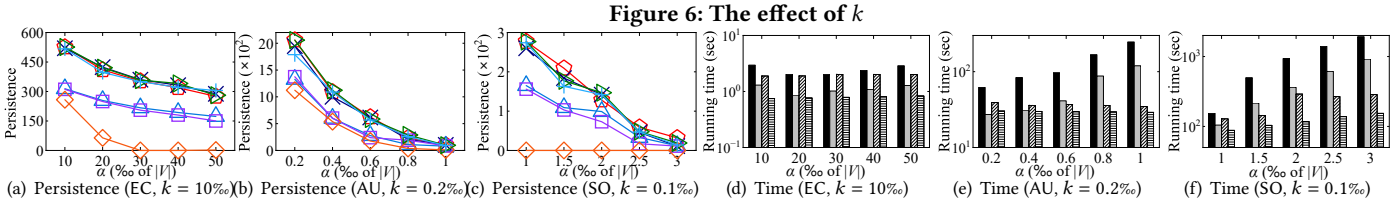
Exp-1: Exact VS. Approximate. Due to the NP-hardness of the InfPM problem, we propose approximate algorithms in Section 4. In this experiment, we compare the exact algorithm Enum with our proposed approximate algorithms RevG and LRep over a small temporal graph EC-Dept3 [28] ($|V|=89$, $|E|=12,216$, $T=803$), with $\alpha = 10\%$ and $\theta = 2$. Table 2 presents the results. We observe that RevG and LRep achieve influence persistence values close to that of Enum, with average approximation ratios of 83.5% and 92.1%, respectively. In terms of efficiency, both RevG and LRep are 3 to 10 orders of magnitude faster than Enum. Notably, Enum failed to complete within a week for $k = 6$, highlighting the practical necessity of our approximate algorithms.

Exp-2: Effect of k. We investigate the impact of the number of seed nodes k (set as ‰ of $|V|$), with results shown in Figure 6. In Figures 6(a)-6(c), RevG, LRep, RevG+, and LRep+ consistently achieve the highest influence persistence, validating the effectiveness of our proposed algorithms. Specifically, compared to the optimal baseline INF, our algorithms achieve up to 100% improvement in total influence persistence. Moreover, influence persistence increases with larger k, as more seeds influence more nodes per snapshot, raising the number of snapshots satisfying threshold α. Figures 6(d)-6(f) show that RevG+ and LRep+ run significantly faster than RevG and LRep, thanks to SCC-DAG-based sampling on compressed graphs. This approach shortens sampling paths and reduces the number of node visits per SCC-RR set, leading to up to a 400% improvement in efficiency over RevG and LRep. Additionally, as k increases, RevG and RevG+ generally run faster, since the greedy deletion process involves fewer iterations. In contrast, LRep and LRep+ exhibit increased running time with larger k, due to a larger initial seed set and more replacements in each iteration. Furthermore, LRep (resp. LRep+) is generally faster than RevG (resp. RevG+) in most cases. This is because LRep (resp. LRep+) prioritizes replacing the weakest seed and terminates early if no improvement in influence persistence is observed. In contrast, RevG (resp. RevG+) exhaustively examines all candidates in each iteration to select one for removal, resulting in a higher computational cost per iteration.



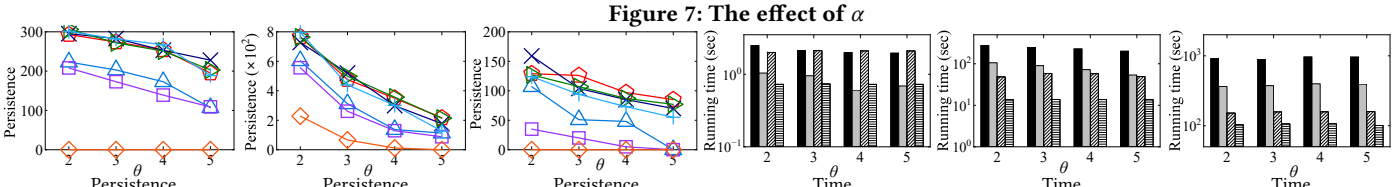
(a) Persistence (EC, $\alpha = 50‰$) (b) Persistence (AU, $\alpha = 1‰$) (c) Persistence (SO, $\alpha = 0.5‰$) (d) Time (EC, $\alpha = 50‰$) (e) Time (AU, $\alpha = 1‰$) (f) Time (SO, $\alpha = 0.5‰$)

Figure 6: The effect of k



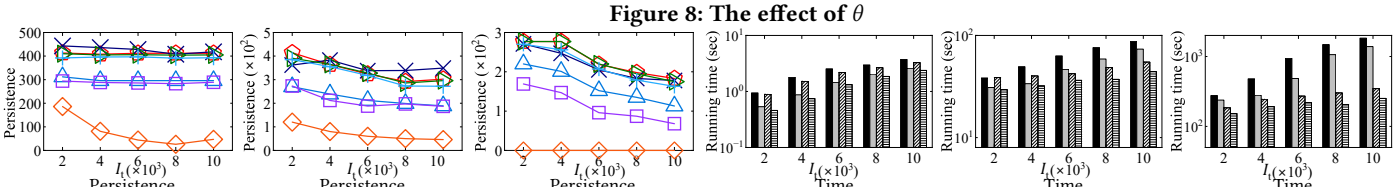
(a) Persistence (EC, $k = 10‰$) (b) Persistence (AU, $k = 0.2‰$) (c) Persistence (SO, $k = 0.1‰$) (d) Time (EC, $k = 10‰$) (e) Time (AU, $k = 0.2‰$) (f) Time (SO, $k = 0.1‰$)

Figure 7: The effect of α



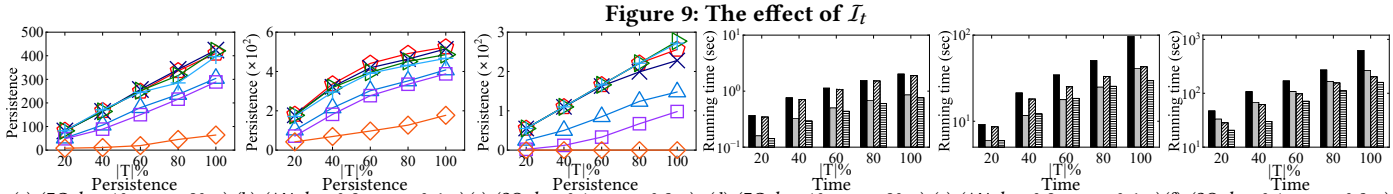
(a) (EC, $k = 20‰$, $\alpha = 50‰$) (b) (AU, $k = 0.5‰$, $\alpha = 1‰$) (c) (SO, $k = 0.2‰$, $\alpha = 0.5‰$) (d) (EC, $k = 20‰$, $\alpha = 50‰$) (e) (AU, $k = 0.5‰$, $\alpha = 1‰$) (f) (SO, $k = 0.2‰$, $\alpha = 0.5‰$)

Figure 8: The effect of θ



(a) (EC, $k = 10‰$, $\alpha = 20‰$) (b) (AU, $k = 0.2‰$, $\alpha = 0.6‰$) (c) (SO, $k = 0.1‰$, $\alpha = 0.3‰$) (d) (EC, $k = 10‰$, $\alpha = 20‰$) (e) (AU, $k = 0.2‰$, $\alpha = 0.6‰$) (f) (SO, $k = 0.1‰$, $\alpha = 0.3‰$)

Figure 9: The effect of $\mathcal{I}_t$



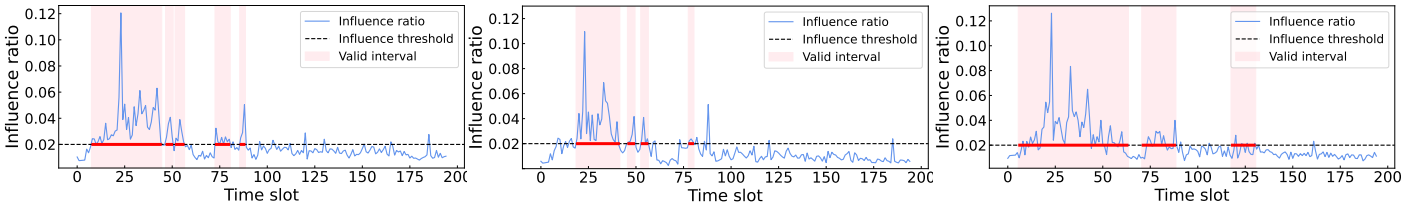
(a) (EC, $k = 10‰$, $\alpha = 20‰$) (b) (AU, $k = 0.2‰$, $\alpha = 0.6‰$) (c) (SO, $k = 0.1‰$, $\alpha = 0.3‰$) (d) (EC, $k = 10‰$, $\alpha = 20‰$) (e) (AU, $k = 0.2‰$, $\alpha = 0.6‰$) (f) (SO, $k = 0.1‰$, $\alpha = 0.3‰$)

Figure 10: The effect of T

Exp-3: Effect of α. We study the impact of the influence threshold α and report the results in Figure 7. Figures 7(a)-7(c) reveal that the influence persistence decreases as α increases. This occurs because, for a fix-sized seed node set, a larger α results in fewer snapshots meeting the threshold, thereby reducing influence persistence. Figures 7(d)-7(f) show that the running times of LRep and LRep+ remain relatively stable on EC and AU, as their runtime is dominated by seed replacement, which primarily depends on the seed budget k. Varying α only affects the computation of persistent intervals, which incurs relatively minor overhead. However, on the larger graph SO, the running times of LRep and LRep+ increase as α grows. This is because higher α values lead to fewer qualifying snapshots, which become more sparsely and discontinuously distributed. This increases the randomness in whether replacing the weakest node improves the influence persistence, thereby leading to more replacements and persistence evaluations. In contrast, the running times of RevG and RevG+ increase with α, because a higher threshold α retains more nodes in the candidate set whenever its size exceeds $\alpha \cdot |V|$. This leads to more iterations in the greedy deletion process, thus increasing the total running time.

Exp-4: Effect of θ. We explore the impact of the persistence threshold θ, with results shown in Figure 8. In Figures 8(a)-8(c), influence persistence decreases as θ grows. This is because, given a fixed set of snapshots meeting threshold α, a higher θ enforces stricter continuity, yielding fewer qualified time intervals and lower overall influence persistence. Figures 8(d)-8(f) show that the running times of LRep and LRep+ remain nearly constant, while those of RevG and RevG+ slightly decrease as θ increases. This occurs because larger θ values cause more snapshots to be pruned during candidate set generation, shrinking the candidate set and reducing the time needed for greedy node deletion, thus lowering total runtime.

Exp-5: Effect of $\mathcal{I}_t$. We evaluate the impact of the number of samples per snapshot $\mathcal{I}_t$. The results are presented in Figure 9. In Figures 9(a)-9(c), it is observed that as $\mathcal{I}_t$ increases, the influence persistence of all algorithms initially decreases and then stabilizes. This is because a larger $\mathcal{I}_t$ enables a more accurate unbiased estimation of influence spread within each snapshot, thereby yielding more consistent influence persistence, as proven by Theorem 7. Furthermore, as shown in Figures 9(d)-9(f), the running times of all algorithms increase with growing $\mathcal{I}_t$.



(a) Persistent intervals achieved by RevG for InfPM (b) Persistent intervals achieved by INF for InfPM (c) Persistent intervals achieved by WRG for Win-InfPM

Figure 11: Influence Persistent intervals in CM

Table 3: Influence Persistence vs. W ($\theta = 5$)

Dataset	CM		EC		MO		AU		SU		WT		SO	
Alg.	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR
$W = 6$	34	28	427	423	413	516	328	362	657	491	818	816	224	210
$W = 7$	38	36	481	476	633	845	565	701	946	1138	995	941	316	256
$W = 8$	47	32	532	549	969	996	1004	984	1537	1330	1032	934	422	310
$W = 9$	49	52	556	561	1225	1280	1327	1246	1647	1643	1053	965	516	387
$W = 10$	53	51	547	546	1460	1560	1739	1484	2277	2194	1105	1004	571	502

Table 4: Running time (sec) vs. W ($\theta = 5$)

Dataset	CM		EC		MO		AU		SU		WT		SO	
Alg.	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR	WRG	WLR
$W = 6$	1.011	0.670	2.512	2.035	42.94	16.14	268.9	51.02	544.7	85.91	297.4	264.5	762.8	235.9
$W = 7$	0.885	0.675	2.601	2.111	41.23	17.83	268.9	50.74	560.8	83.8	215.1	206.5	593.3	231.1
$W = 8$	0.756	0.668	3.029	2.377	37.23	16.89	268.6	47.56	551.9	78.06	151.7	213.2	484.8	204.4
$W = 9$	0.911	0.727	3.018	2.146	41.47	15.55	271.6	45.67	549.1	83.84	156.7	202.9	397.6	217.3
$W = 10$	0.888	0.671	2.977	2.134	41.35	16.61	242.7	47.65	484.8	84.47	114.6	179.5	355.7	211.2

Exp-6: Effect of T. We study the scalability of our algorithms by varying the time span T, i.e., the number of snapshots. To this end, we retain different fractions of snapshots from the entire temporal graph to generate temporal graphs with different Ts. The results are reported in Figure 10. As expected, both the influence persistence and running time increase when the time span grows. This is because more snapshots lead to more qualified persistent intervals and longer processing time for computing the influence spread across all snapshots.

7.3 Evaluation of Win-InfPM Algorithm

In this section, we assess the performance of the WRG and WLR algorithms designed for the Win-InfPM problem. Derived from RevG+ and LRep+ respectively, both integrate our proposed PssW procedure to compute window-based influence persistence.

Exp-7: Effect of W. We evaluate the effect of the window size W while fixing the persistence threshold at the default $\theta = 5$. As shown in Table 3, influence persistence for WRG and WLR increases as W grows. The reason behind is that, a larger window size enhances the temporal continuity flexibility, allowing snapshots that exceed the influence threshold and occur close in time, even if not strictly consecutive, to be counted together within a persistent interval. Consequently, a greater number of intervals satisfy the persistence condition, and longer qualifying intervals are more likely to emerge, improving the overall influence persistence. Furthermore, Table 4 shows the running time of WRG and WLR w.r.t. different W. On smaller datasets, both algorithms exhibit relatively stable runtime with minimal sensitivity to window size. However, on larger datasets like WT and SO, both WRG and WLR run faster when W increases. This is because the increased window size produces fewer but longer qualifying intervals, which can be constructed through simple extensions, lowering the overhead of re-evaluating each timestamp for interval validity.

7.4 Case Study

We conduct a case study on the real-world temporal social network CollegeMsg, which records message exchanges among university users over time. We consider scenarios such as campus-wide event promotion or sustained visibility of safety alerts, where maintaining long-term user engagement is the key objective. To model this, we define a persistent interval as a period during which the number of influenced users exceeds a threshold (i.e., $\alpha = 20‰$) for at least $\theta = 3$ consecutive days. To this end, the goal is to select a small set of seed users (i.e., $k = 10‰$) to maximize the total duration of persistent intervals. We visualize three cases to illustrate persistent intervals under different actions taken by the college.

Figure 11(a) shows the persistent intervals achieved by RevG, where the influence threshold is exceeded for at least 3 consecutive days (i.e., InfPM). Most time slots that exceed the influence threshold fall within the 0–100 range, yielding several durations that satisfy the consecutive requirement. Due to space constraints, we only present results for RevG, as LRep yields similar outcomes. Figure 11(b) presents the persistent intervals produced by the optimal baseline INF method. Compared to RevG, INF produces fewer and shorter persistent intervals, leading to significantly lower overall influence persistence. Figure 11(c) displays the persistent intervals identified by WRG, where the influence threshold is exceeded on at least 3 days within each one-week window ($W = 7$) (i.e., Win-InfPM). This added flexibility enables WRG to include time slots that individually fall short of the threshold but collectively contribute to sustained influence within the window, yielding longer intervals and higher overall influence persistence than RevG.

8 Conclusion

In this paper, we introduce the novel Influence Persistence Maximization (InfPM) problem, which aims to maximize influence persistence across snapshots in temporal social networks. Due to the NP-hardness of InfPM, we develop two seed selection algorithms: RevG, which employs a reverse deletion greedy strategy, and LRep, which utilizes a replacement-based strategy. To improve the efficiency of RevG and LRep, we propose a new influence computation method that compresses each snapshot into a compact directed acyclic graph, and integrates a novel sampling technique with an unbiased influence estimator that provides theoretical guarantees. Furthermore, we study a practical variant Win-InfPM and effectively solve it. Extensive experiments on real-world datasets demonstrate that our proposed algorithms are effective, efficient, and scalable. In future work, we plan to explore real-time influence persistence maximization over streaming temporal networks, where snapshots arrive continuously in a streaming manner.

References

[1] Zahra Aghaee, Mohammad Mahdi Ghasemi, Hamid Ahmadi Beni, Asgarali Bouyer, and Afsaneh Fatemi. 2021. A survey on meta-heuristic algorithms for the influence maximization problem in the social networks. *Computing* 103 (2021), 2437–2477.

[2] CDC Archive. 2022. Ways Health Departments Can Help Increase COVID-19 Vaccinations. Retrieved July 1, 2025 from https://archive.cdc.gov/www_cdc_gov/vaccines/covid-19/health-departments/generate-vaccinations.html?

[3] Michael O Ball. 1980. Complexity of network reliability computations. *Networks* 10, 2 (1980), 153–165.

[4] Prithu Banerjee, Wei Chen, and Laks VS Lakshmanan. 2019. Maximizing welfare in social networks under a utility driven influence diffusion model. In *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*. 1078–1095.

[5] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. 2014. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 946–957.

[6] Builtin. 2023. How the Mere Exposure Effect Works in Marketing and Advertising. Retrieved July 1, 2025 from https://builtin.com/articles/mere-exposure-effect

[7] Xueqin Chang, Jiajie Fu, Qing Liu, Yunjun Gao, and Baihua Zheng. 2025. Time-Aware Influence Minimization via Blocking Social Networks. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 557–570.

[8] Xueqin Chang, Xiangyu Ke, Lu Chen, Congcong Ge, Ziheng Wei, and Yunjun Gao. 2023. Host profit maximization: Leveraging performance incentives and user flexibility. *Proceedings of the VLDB Endowment* 17, 1 (2023), 51–64.

[9] Xueqin Chang, Qing Liu, Yunjun Gao, Baihua Zheng, Yi Cai, and Qing Li. 2025. The Most Influenced Community Search on Social Networks. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2601–2614.

[10] Xueqin Chang, Qing Liu, Baihua Zheng, and Yunjun Gao. 2025. Influence Persistence Maximization in Temporal Social Networks. https://github.com/ZJU-DAILY/INFPM

[11] 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*. 1029–1038.

[12] Benjamin Denison, Heather Dahlen, Jae-Eun C Kim, Christopher Williams, Elissa Kranzler, Joseph N Luchman, Sarah Trigger, Morgane Bennett, Tyler Nighbor, Monica Vines, et al. 2023. Evaluation of the "we can do this" campaign paid media and COVID-19 vaccination uptake, United States, December 2020–January 2022. *Journal of health communication* 28, 9 (2023), 573–584.

[13] Pedro Domingos and Matt Richardson. 2001. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 57–66.

[14] Keywords Everywhere. 2024. 55 Facebook Stats For Your Social Media Strategy [2024]. Retrieved July 1, 2025 from https://keywordseverywhere.com/blog/facebook-stats/

[15] Chen Feng, Xingguang Chen, Qintian Guo, Fangyuan Zhang, and Sibo Wang. 2024. Efficient Approximation Algorithms for Minimum Cost Seed Selection with Probabilistic Coverage Guarantee. *Proceedings of the ACM on Management of Data* 2, 4 (2024), 1–26.

[16] Qintian Guo, Chen Feng, Fangyuan Zhang, and Sibo Wang. 2023. Efficient algorithm for budgeted adaptive influence maximization: An incremental RR-set update approach. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–26.

[17] Qintian Guo, Sibo Wang, Zhewei Wei, Wenqing Lin, and Jing Tang. 2022. Influence Maximization Revisited: Efficient Sampling with Bound Tightened. *ACM Transactions on Database Systems (TODS)* (2022).

[18] Kai Han, Benwei Wu, Jing Tang, Shuang Cui, Cigdem Aslay, and Laks VS Lakshmanan. 2021. Efficient and effective algorithms for revenue maximization in social advertising. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*. 671–684.

[19] Keke Huang, Sibo Wang, Glenn Bevilacqua, Xiaokui Xiao, and Laks VS Lakshmanan. 2017. Revisiting the stop-and-stare algorithms for influence maximization. *Proceedings of the VLDB Endowment* 10, 9 (2017), 913–924.

[20] Shixun Huang, Zhifeng Bao, J Shane Culpepper, and Bang Zhang. 2019. Finding temporal influential users over evolving social networks. In *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 398–409.

[21] Yiqian Huang, Shiqi Zhang, Laks VS Lakshmanan, Wenqing Lin, Xiaokui Xiao, and Bo Tang. 2024. Efficient and Effective Algorithms for A Family of Influence Maximization Problems with A Matroid Constraint. *Proceedings of the VLDB Endowment* 18, 2 (2024), 117–129.

[22] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 137–146.

[23] Kim Kovelle. 2025. Short-Term vs. Long-Term Digital Marketing Campaigns. Retrieved July 1, 2025 from https://thezoeteam.com/blog/short-term-vs-long-term-marketing

[24] Later. 2025. Instagram Influencer Marketing: Master Guide for Brands in 2025. Retrieved July 1, 2025 from https://later.com/blog/instagram-influencer-marketing/

[25] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 462–470.

[26] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.

[27] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 420–429.

[28] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[29] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 797–808.

[30] Weihua Li, Quan Bai, Minjie Zhang, and Tung Doan Nguyen. 2018. Automated influence maintenance in social networks: an agent-based approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 31, 10 (2018), 1884–1897.

[31] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 30, 10 (2018), 1852–1872.

[32] Zhicheng Liang, Yu Yang, Xiangyu Ke, Xiaokui Xiao, and Yunjun Gao. 2024. A Benchmark Study of Deep-RL Methods for Maximum Coverage Problems over Graphs. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3666–3679.

[33] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

[34] Bastian Moritz. 2024. The Strategic Use of Repeated Exposure and the Phenomenon of Familiarity. Retrieved July 1, 2025 from https://www.mxmoritz.com/article/the-mere-exposure-effect-in-marketing-and-sales

[35] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. 2016. Dynamic influence analysis in evolving networks. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1077–1088.

[36] Naoto Ohsaka, Tomohiro Sonobe, Sumio Fujita, and Ken-ichi Kawarabayashi. 2017. Coarsening massive influence networks for scalable diffusion analysis. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*. 635–650.

[37] Openinfluence. 2025. YouTube Influencer Marketing. Retrieved July 1, 2025 from https://openinfluence.com/youtube-influencer-marketing/

[38] Shopify. 2025. Influencer Pricing: The Cost of Influencers in 2025. Retrieved July 1, 2025 from https://www.shopify.com/sg/blog/influencer-pricing

[39] Guojie Song, Yuanhao Li, Xiaodong Chen, Xinran He, and Jie Tang. 2016. Influential node tracking on dynamic social network: An interchange greedy approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 29, 2 (2016), 359–372.

[40] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. 2018. Online processing algorithms for influence maximization. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*. 991–1005.

[41] Jing Tang, Xueyan Tang, and Junsong Yuan. 2017. Profit maximization for viral marketing in online social networks: Algorithms and analysis. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 30, 6 (2017), 1095–1108.

[42] 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*. 1539–1554.

[43] 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*. 75–86.

[44] Leslie G Valiant. 1979. The complexity of enumeration and reliability problems. *siam Journal on Computing* 8, 3 (1979), 410–421.

[45] Jinghao Wang, Yanping Wu, Xiaoyang Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2024. Efficient Influence Minimization via Node Blocking. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2501–2513.

[46] Yanhao Wang, Qi Fan, Yuchen Li, and Kian-Lee Tan. 2017. Real-time influence maximization on dynamic social streams. *Proceedings of the VLDB Endowment* 10, 7 (2017), 805–816.

[47] Xudong Wu, Luoyi Fu, Zixin Zhang, Huan Long, Jingfan Meng, Xinbing Wang, and Guihai Chen. 2020. Evolving influence maximization in evolving networks. *ACM Transactions on Internet Technology (TOIT)* 20, 4 (2020), 1–31.

[48] Jiadong Xie, Zehua Chen, Deming Chu, Fan Zhang, Xuemin Lin, and Zhihong Tian. 2024. Influence maximization via vertex countering. *Proceedings of the VLDB Endowment* 17, 6 (2024), 1297–1309.

[49] Vijaya Krishna Yalavarthi and Arijit Khan. 2018. Steering top-k influencers in dynamic graphs via local updates. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 576–583.
[50] Yu Yang and Jian Pei. 2019. Influence analysis in evolving networks: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 33, 3 (2019), 1045–1063.
[51] Yu Yang, Zhefeng Wang, Tianyuan Jin, Jian Pei, and Enhong Chen. 2019. Tracking top-k influential users with relative errors. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1783–1792.
[52] Yu Yang, Zhefeng Wang, Jian Pei, and Enhong Chen. 2017. Tracking influential individuals in dynamic networks. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 29, 11 (2017), 2615–2628.
[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*. 671–680.
[54] Junzhou Zhao, Shuo Shang, Pinghui Wang, John CS Lui, and Xiangliang Zhang. 2019. Tracking influential nodes in time-decaying dynamic interaction networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1106–1117.
[55] Junzhou Zhao, Pinghui Wang, Wei Zhang, Zhaosong Zhang, Maoli Liu, Jing Tao, and John CS Lui. 2023. Tracking Influencers in Decaying Social Activity Streams With Theoretical Guarantees. *IEEE/ACM Transactions on Networking* 32, 2 (2023), 1461–1476.

Appendix

A Proof of Theorem 4

Proof. We prove by contradiction. Suppose snapshot G_t cannot be excluded, implying there exists an optimal seed set S^* that needs G_t to achieve maximum influence persistence. For G_t to contribute to the persistence of S^*, we must have $\sigma_t(S^*) \geq \alpha \cdot |V|$ (by the α-valid condition). However, the theorem premise states that $\max_{S\subseteq V, |S|=k} \sigma_t(S) < \alpha \cdot |V|$, which means $\sigma_t(S^*) < \alpha \cdot |V|$. This contradicts the α-valid requirement. Therefore, G_t can be safely excluded without affecting optimality. □

B Proof of Theorem 5

Proof. We prove by contradiction. Suppose snapshot G_t with $t \notin T_\theta$ cannot be excluded, implying there exists an optimal seed set S^* that requires G_t to achieve maximum influence persistence. For G_t to contribute to the persistence of S^*, it must be part of a consecutive temporal interval of length at least θ (by the θ-persistent condition). However, by definition of T_θ, snapshot G_t is not contained in any persistent influence interval of length $\geq \theta$ when applying GetPersistence to the α-valid snapshots T_α. This means G_t cannot satisfy the θ-persistent requirement for any feasible solution. Thus, G_t can be safely excluded without affecting optimality. □

C Proof of Theorem 6

Proof. We prove that if a timestamp $t \notin \bigcup_{I\in\mathcal{T}(A,\mathcal{G})} I$, then for any $A' \subset A$, $t \notin \bigcup_{I'\in\mathcal{T}(A',\mathcal{G})} I'$. Suppose for contradiction that $\exists A' \subset A$ such that $t \in \bigcup_{I'\in\mathcal{T}(A',\mathcal{G})} I'$. Then $\exists I' = [t_s, t_e] \in \mathcal{T}(A', \mathcal{G})$ with $t \in I'$ that is a maximal (α, θ, A')-persistent influence interval. Since $A' \subset A$ and $\sigma_t(\cdot)$ is monotonic, we have $\sigma_{t'}(A) \geq \sigma_{t'}(A') \geq \alpha \cdot |V|$ for all $t' \in I'$. Thus I' satisfies both α-valid and θ-persistent for A. By Definition 1, I' must be contained in some maximal (α, θ, A)-persistent influence interval $I \in \mathcal{T}(A, \mathcal{G})$. Since $t \in I' \subseteq I$, we have $t \in \bigcup_{I\in\mathcal{T}(A,\mathcal{G})} I$, contradicting our assumption. Therefore, $\forall A' \subset A$, $t \notin \bigcup_{I'\in\mathcal{T}(A',\mathcal{G})} I'$. □

D Proof of Theorem 7

Proof. We prove it by extending chernoff inequalities [33]. Applying chernoff inequalities, we have $\Pr[f_{\tilde{\mathcal{R}}_t}(S)\cdot|\tilde{\mathcal{R}}_t| - |\tilde{\mathcal{R}}_t|\cdot\tilde{\sigma}_t(S) \geq \epsilon\cdot|\tilde{\mathcal{R}}_t|\cdot\tilde{\sigma}_t(S)] \leq \exp\left(-\frac{\epsilon^2}{2+\epsilon}\cdot|\tilde{\mathcal{R}}_t|\cdot\tilde{\sigma}_t(S)\right)$. When $|\tilde{\mathcal{R}}_t| \geq \frac{(2+\epsilon)\cdot\log\delta}{\epsilon^2\cdot\tilde{\sigma}_t(S)}$, we have $\Pr[f_{\tilde{\mathcal{R}}_t}(S) - \tilde{\sigma}_t(S) \geq \epsilon\cdot\tilde{\sigma}_t(S)] \leq \delta$. Thus, $|f_{\tilde{\mathcal{R}}_t}(S) - \tilde{\sigma}_t(S)| < \epsilon\cdot\tilde{\sigma}_t(S)$ holds with probability at least $1-\delta$. □

E Proof of Lemma 1

Proof. In the compressed graph D_t, nodes within the same SCC C_j can influence each other with weight $\tilde{w}_t^{in}(u,v) = 1$, while inter-component influence is given by $w_t^{out}(u,v) = p_t(C_i, C_j)$ for $u \in C_i, v \in C_j$. Then, we have $\tilde{w}_t^{in}(u,v) = 1 \geq w_t(u,v)$ and $w_t^{out}(u,v) = p_t(C_i,C_j) = 1 - \prod_{(u,v)\in E_t,\ u\in C_i,\ v\in C_j}(1 - w_t(u,v)) \geq w_t(u,v)$. Compared to the original graph G_t, where influence follow actual paths and edge activations, D_t assumes maximal intra-SCC influence and abstracts inter-SCC influence probabilistically. Therefore, for any seed set $S \subseteq V$, the estimated influence $\tilde{\sigma}_t(S)$ in D_t is at least the true influence $\sigma_t(S)$ in G_t. □

F Proof of Lemma 2

Proof. Let the edge set E_t be partitioned into disjoint subsets $E_t^1, \ldots, E_t^\ell$ and $E_t^0 = E_t \setminus \bigcup_{j=1}^{\ell} E_t^j$, where each E_t^j contains all intra-cluster edges within node subset $C_j \subseteq V$, and E_t^0 contains the remaining inter-cluster edges. For each $j \in [\ell]$, define the strongly connected reliability of the subgraph $G_t[C_j]$ as: $\mathrm{Scr}(G_t[C_j]) = \sum_{X\subseteq E_t^j} p(X \mid E_t^j)\cdot \mathbf{1}\left[(C_j, X) \text{ is strongly connected (SC)}\right]$ [36]. Let $\mathcal{E}_j \subseteq E_t^j$ and $\mathcal{E}_0 \subseteq E_t^0$ be the random edge subsets sampled under the probabilistic model. Define $\mathcal{E} = \bigcup_{j=1}^{\ell}\mathcal{E}_j \cup \mathcal{E}_0$ and $E_t' = \bigcup_{j=1}^{\ell} E_t^j \cup \mathcal{E}_0$). $r(V,\mathcal{E})(S) = r(V, E_t')(S)$ when every subgraph $(C_j, \mathcal{E}_j)$ is strongly connected for all $j \in [\ell]$. Thus,

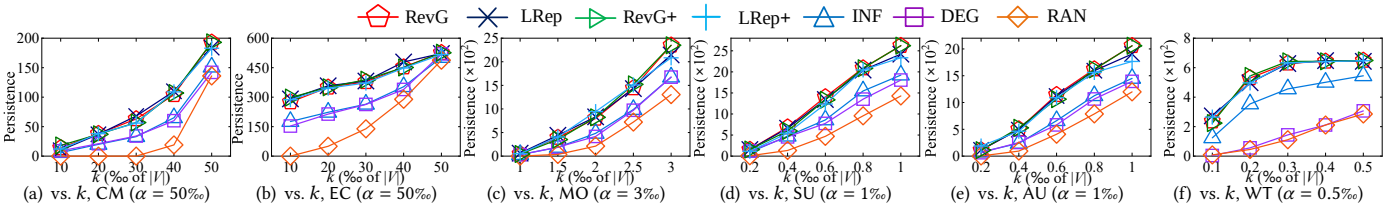
$$
\begin{aligned}
\sigma_t(S) &= \sum_{\mathcal{E}_1\subseteq E_1}\cdots\sum_{\mathcal{E}_\ell\subseteq E_\ell}\left(\prod_{j=1}^{\ell} p(\mathcal{E}_j \mid E_t^j)\right)\sum_{\mathcal{E}_0\subseteq E_0} p(\mathcal{E}_0 \mid E_t^0)\cdot r(V,\mathcal{E})(S) \\
&\geq \sum_{\mathcal{E}_1\subseteq E_1}\cdots\sum_{\mathcal{E}_\ell\subseteq E_\ell}\left(\prod_{j=1}^{\ell} p(\mathcal{E}_j \mid E_t^j)\cdot \mathbf{1}[(C_j,\mathcal{E}_j) \text{ is SC}]\right) \\
&\quad\cdot \sum_{\mathcal{E}_0\subseteq E_0} p(\mathcal{E}_0 \mid E_t^0)\cdot r(V, E_t')(S) \\
&= \left(\prod_{j=1}^{\ell}\mathrm{Scr}(G_t[C_j])\right)\cdot\tilde{\sigma}_t(S) \qquad (10)
\end{aligned}
$$

where $\tilde{\sigma}_t(S)$ is the expected influence under the condition that all $G_t[C_j]$ are strongly connected. Therefore, we have $\tilde{\sigma}_t(S) \leq \left(\prod_{j=1}^{\ell}\mathrm{Scr}(G_t[C_j])^{-1}\right)\cdot\sigma_t(S)$. □

G Full experiment results

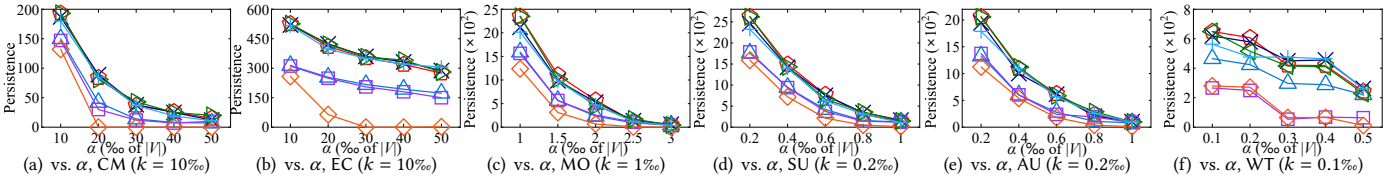
This section shows the complete experiment results that are omitted in Section IV due to space constraints.

Effect of k. We investigate the impact of the number of seed nodes k (set as ‰ of $|V|$). In Figures 12 and 16, RevG, LRep, RevG+, and LRep+ consistently achieve the highest influence persistence, validating the effectiveness of our proposed algorithms. Specifically, compared to the optimal baseline INF, our algorithms achieve up to 100% improvement in total influence persistence. Moreover, influence persistence increases with larger k, as more seeds influence more nodes per snapshot, raising the number of snapshots satisfying threshold α. Figures 17 and 21 show that RevG+ and LRep+ consistently outperform RevG and LRep, benefiting from SCC-DAG-based sampling on compressed graphs, which shortens sampling paths and reduces node visits per SCC-RR set, leading to up to a 400% improvement in efficiency over RevG and LRep. Additionally, as k increases, RevG and RevG+ generally run faster, needing fewer iterations in the greedy deletion process. In contrast, LRep and LRep+ exhibit increased running time with larger k, due to larger initial seed set and more replacements. Furthermore, LRep (resp. LRep+) is generally faster than RevG (resp. RevG+) in most cases, because RevG (resp. RevG+) examines all candidates per iteration to identify one for removal, while LRep (resp. LRep+) only examines the current seed set for replacement. As the candidate set is typically larger, LRep (resp. LRep+) incurs lower computational cost per iteration. For small dataset like EC, when k is small, the running time of RevG and RevG+ is dominated by influence estimation and



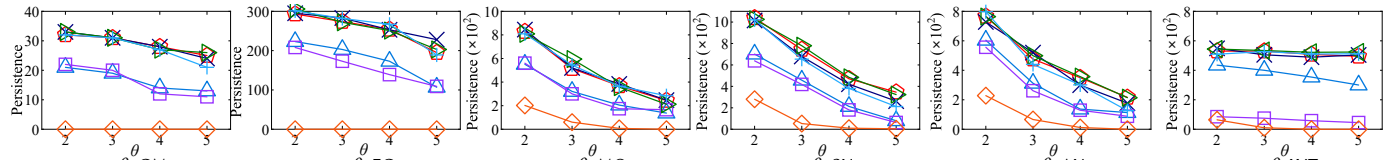
(a) vs. k, CM (α = 50‰) (b) vs. k, EC (α = 50‰) (c) vs. k, MO (α = 3‰) (d) vs. k, SU (α = 1‰) (e) vs. k, AU (α = 1‰) (f) vs. k, WT (α = 0.5‰)

Figure 12: Influence Persistence vs. k



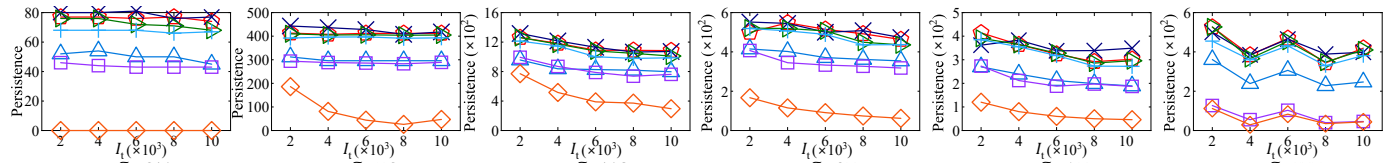
(a) vs. α, CM (k = 10‰) (b) vs. α, EC (k = 10‰) (c) vs. α, MO (k = 1‰) (d) vs. α, SU (k = 0.2‰) (e) vs. α, AU (k = 0.2‰) (f) vs. α, WT (k = 0.1‰)

Figure 13: Influence Persistence vs. α



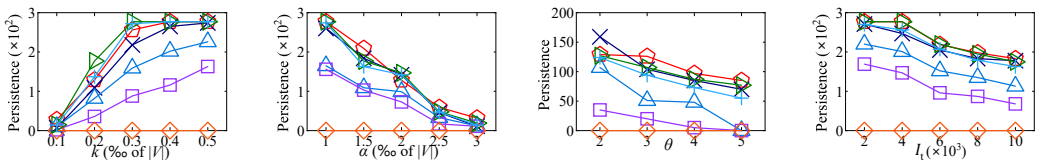
(a) vs. θ, CM (k = 20‰, α = 50‰) (b) vs. θ, EC (k = 20‰, α = 50‰) (c) vs. θ, MO (k = 2‰, α = 3‰) (d) vs. θ, SU (k = 0.5‰, α = 1‰) (e) vs. θ, AU (k = 0.5‰, α = 1‰) (f) vs. θ, WT (k = 0.2‰, α = 0.5‰)

Figure 14: Influence Persistence vs. θ



(a) vs. $\mathcal{I}_t$, CM (k = 10‰, α = 20‰) (b) vs. $\mathcal{I}_t$, EC (k = 10‰, α = 20‰) (c) vs. $\mathcal{I}_t$, MO (k = 1‰, α = 1.5‰) (d) vs. $\mathcal{I}_t$, SU (k = 0.2‰, α = 0.6‰) (e) vs. $\mathcal{I}_t$, AU (k = 0.2‰, α = 0.6‰) (f) vs. $\mathcal{I}_t$, WT (k = 0.1‰, α = 0.3‰)

Figure 15: Influence Persistence vs. $\mathcal{I}_t$



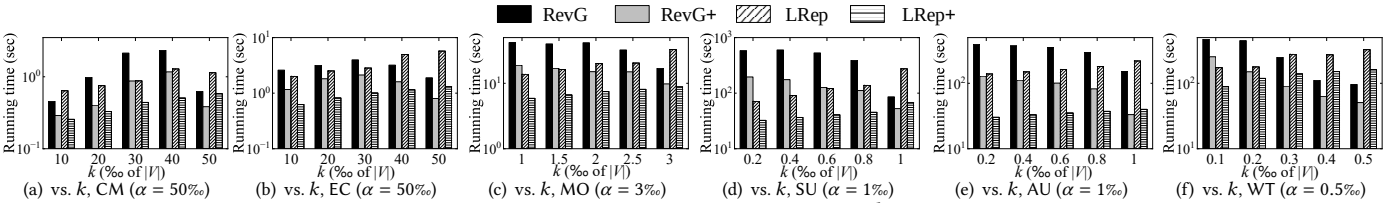
(a) vs. k, (α = 2‰) (b) vs. α, (k = 0.2‰) (c) vs. θ, (k = 0.2‰, α = 2‰) (d) vs. $\mathcal{I}_t$, (k = 0.3‰, α = 2‰)

Figure 16: Influence Persistence on SO

persistence computation, both of which become more expensive with k. When k is sufficiently large, fewer candidates are deleted, reducing overall runtime.

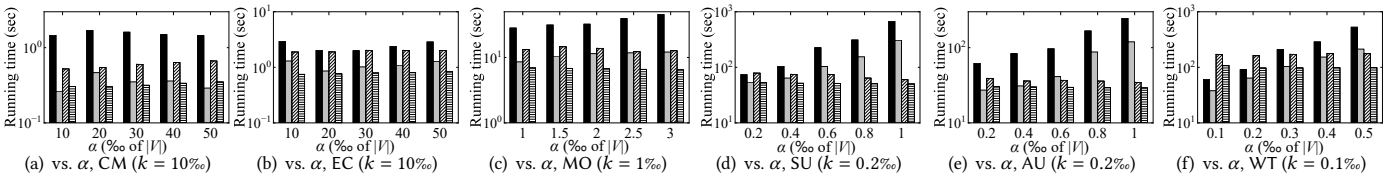
Effect of α. We study the impact of the influence threshold α. Figures 13 and 16 reveal that the influence persistence decreases as α increases. This occurs because, for a fix-sized seed node set, a larger α results in fewer snapshots meeting the threshold, thereby reducing influence persistence. Figures 18 and 21 show that the running times of LRep and LRep+ remain relatively stable on EC and AU, as their runtime is dominated by seed replacement, which primarily depends on the seed budget k. Varying α only affects the computation of persistent intervals, which incurs relatively minor overhead. However, on the larger graph SO, the running times of LRep and LRep+ increase as α grows. This is because higher α values lead to fewer qualifying snapshots, which become more sparsely and discontinuously distributed. This increases the randomness in whether replacing the weakest node improves the influence persistence, thereby leading to more replacements and persistence evaluations. In contrast, the running times of RevG and RevG+ increase with α, because a higher threshold α retains more nodes in the candidate set whenever its size exceeds $\alpha \cdot |V|$. This leads to more iterations in the greedy deletion process, thus increasing the total running time.

Effect of θ. We explore the impact of the persistence threshold θ. In Figures 14 and 16, influence persistence decreases as θ grows. This is because, given a fixed set of snapshots meeting threshold α, a higher θ enforces stricter continuity, yielding fewer qualified time intervals and lower overall influence persistence. Figures 19 and 21 show that the running times of LRep and LRep+ remain nearly constant, while those of RevG and RevG+ slightly decrease as θ increases. This occurs because larger θ values cause more snapshots to be pruned during candidate set generation, shrinking



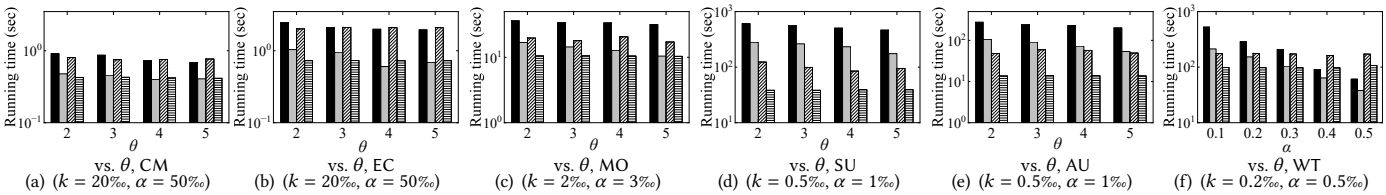
(a) vs. k, CM ($\alpha = 50‰$) (b) vs. k, EC ($\alpha = 50‰$) (c) vs. k, MO ($\alpha = 3‰$) (d) vs. k, SU ($\alpha = 1‰$) (e) vs. k, AU ($\alpha = 1‰$) (f) vs. k, WT ($\alpha = 0.5‰$)

Figure 17: Running time vs. k



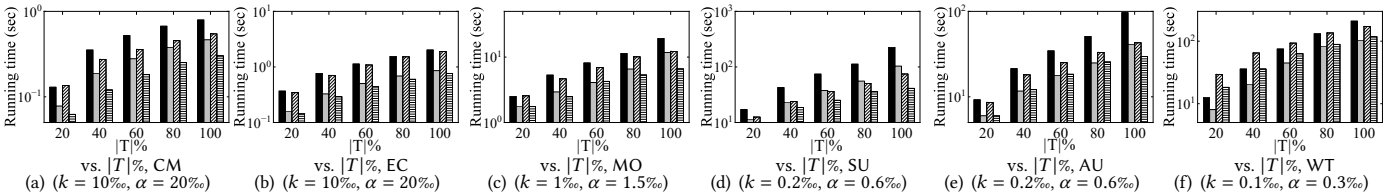
(a) vs. α, CM ($k = 10‰$) (b) vs. α, EC ($k = 10‰$) (c) vs. α, MO ($k = 1‰$) (d) vs. α, SU ($k = 0.2‰$) (e) vs. α, AU ($k = 0.2‰$) (f) vs. α, WT ($k = 0.1‰$)

Figure 18: Running time vs. α



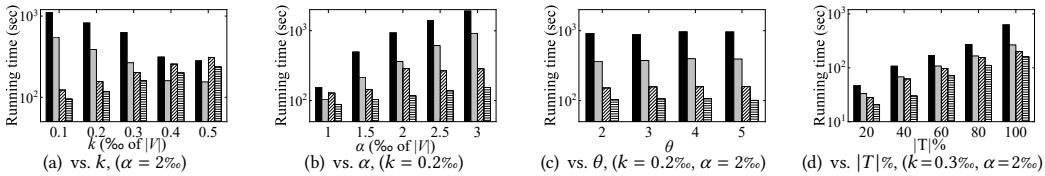
(a) vs. θ, CM ($k = 20‰$, $\alpha = 50‰$) (b) vs. θ, EC ($k = 20‰$, $\alpha = 50‰$) (c) vs. θ, MO ($k = 2‰$, $\alpha = 3‰$) (d) vs. θ, SU ($k = 0.5‰$, $\alpha = 1‰$) (e) vs. θ, AU ($k = 0.5‰$, $\alpha = 1‰$) (f) vs. θ, WT ($k = 0.2‰$, $\alpha = 0.5‰$)

Figure 19: Running time vs. θ



(a) vs. $|T|$%, CM ($k = 10‰$, $\alpha = 20‰$) (b) vs. $|T|$%, EC ($k = 10‰$, $\alpha = 20‰$) (c) vs. $|T|$%, MO ($k = 1‰$, $\alpha = 1.5‰$) (d) vs. $|T|$%, SU ($k = 0.2‰$, $\alpha = 0.6‰$) (e) vs. $|T|$%, AU ($k = 0.2‰$, $\alpha = 0.6‰$) (f) vs. $|T|$%, WT ($k = 0.1‰$, $\alpha = 0.3‰$)

Figure 20: Scalability test



(a) vs. k, ($\alpha = 2‰$) (b) vs. α, ($k = 0.2‰$) (c) vs. θ, ($k = 0.2‰$, $\alpha = 2‰$) (d) vs. $|T|$%, ($k = 0.3‰$, $\alpha = 2‰$)

Figure 21: Running time on SO

the candidate set and reducing the time needed for greedy node deletion, thus lowering total runtime.

Effect of $\mathcal{I}_t$. We evaluate the impact of the number of samples per snapshot $\mathcal{I}_t$. The results are presented in Figure 9. In Figures 15 and 16, it is observed that as $\mathcal{I}_t$ increases, the influence persistence of all algorithms initially decreases and then stabilizes. This is because a larger $\mathcal{I}_t$ enables a more accurate unbiased estimation of influence spread within each snapshot, thereby yielding more consistent influence persistence, as proven by Theorem 7.

Effect of T. We study the scalability of our algorithms by varying the time span T, i.e., the number of snapshots. To this end, we retain different fractions of snapshots from the entire temporal graph to generate temporal graphs with different Ts. The results are reported in Figures 20-21. As expected, running time increase when the time span grows. This is because more snapshots lead to more qualified persistent intervals and longer processing time for computing the influence spread across all snapshots.