Finding Temporal Influential Users over Evolving Social Networks

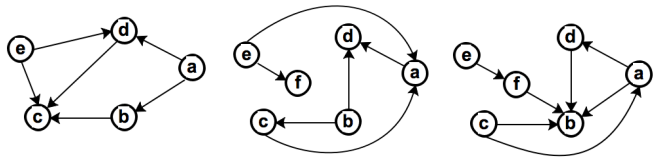
Shixun Huang †, Zhifeng Bao †, J.Shane Culpepper †, Bang Zhang ‡
†*RMIT University, Melbourne, Australia* ‡*Damo Academy Alibaba, Hangzhou, China*
†{shixun.huang, zhifeng.bao, shane.culpepper}@rmit.edu.au, ‡zhangbang.zb@alibaba-inc.com

***Abstract*—Influence maximization (IM) continues to be a key research problem in social networks. The goal is to find a small seed set of target users that have the greatest influence in the network under various stochastic diffusion models. While significant progress has been made on the IM problem in recent years, several interesting challenges remain. For example, social networks in reality are constantly evolving, and "important" users with the most influence also change over time. As a result, several recent studies have proposed approaches to update the seed set as the social networks evolve. However, this seed set is not guaranteed to be the best seed set over a period of time. In this paper we study the problem of *Distinct Influence Maximization (DIM)* where the goal is to identify a seed set of influencers who maximize the number of distinct users influenced over a predefined window of time. Our new approach allows social network providers to make fewer incremental changes to targeted advertising while still maximizing the coverage of the advertisements. It also provides finer grained control over service level agreements where a certain number of impressions for an advertisement must be displayed in a specific time period. We propose two different strategies HCS and VCS with novel graph compression techniques to solve this problem. Additionally, VCS can also be applied directly to the traditional IM problem. Extensive experiments on real-world datasets verify the efficiency, accuracy and scalability of our solutions on both the DIM and IM problems.**

I. Introduction

Social networks are an increasingly important part of our daily lives, and an increasingly important source of behavioural data for research purposes. Understanding how information spreads through a social network is a research problem that has received a great deal of attention in the last decade. Understanding this processes can provide insights in human social structure, which can in turn be leveraged to provide better targeted advertising. One popular strategy for viral marketing is Influence Maximization (IM) [1], which is the problem of finding a ***seed set*** of k target users such that the expected number of users influenced by the target users is maximized under a stochastic diffusion model.

The IM problem is known to be NP-hard [2], and many approximation algorithms and heuristic methods have been proposed over the years to tackle it [2]–[16]. These methods obtain solutions based on the assumption that influence patterns such as relationships among users are static. However, in reality, social networks are constantly evolving over time [17], [18]. Therefore, there are recent studies focusing on how to support iteratively updating the seed set on a *dynamic social network* [19]–[21].



(a) Snapshot1 (b) Snapshot2 (c) Snapshot3

Fig. 1: Three different stages of a social network.

Nevertheless, there are still important limitations in targeted advertising which have not been considered by previous literature. For example, several issues arise as the seed set is updated in an evolving social network. (1) Users may be repeatedly influenced by the updated seed set while the number of distinct users who are influenced might be limited, which constrains the effective user coverage of the influence spread. (2) Advertising (ad) personalization is an important marketing strategy beneficial for both customers and companies [22]. Personalizing advertising messages is a resource-intensive but important process for increasing revenues [23], customer retention [24] and brand loyalty [25]. Due to the dynamic nature of social networks, it may not be realistic to obtain a seed set, redesign personalized advertising messages and deploy them before the social network changes again over a short period of time. (3) Repeatedly changing the seed set does not maximize the *effective frequency* (the frequency level of ad exposures required to drive consumer action) which is crucial in marketing campaigns [26].

To mitigate these issues, we study the distinct influence maximization (DIM) problem which aims to find a *fixed* seed set of k target users to maximize the expected number of distinct users influenced by the target users in an evolving social network. Figure 1 is a simple example which contains three snapshots, each representing a different stage of change (for example three hours, three days, or three weeks) in a social network. In this example, we say a target user x can influence a user y if x can reach y in the graph (x can influence itself as well). Note that influence spread estimation under the diffusion model used in this paper makes the problem more complex. Suppose we are required to find one target user. Previous studies would select different users a, b and c as the target user for the three snapshots since they can influence the maximum

[1]Zhifeng Bao is the corresponding author.

number of users in that snapshot. More specifically, the same set of users $\{a, b, c, d\}$ are repeatedly influenced by the target users a, b and c in different snapshots while users e and f are not influenced. In contrast, the DIM problem aims to find a common set of target users among these snapshots to maximize the number of distinct users *across all of the snapshots* who can be influenced by the common target users. For this case, we would select user e as the target user since e can influence all users $\{a, b, c, d, e, f\}$ even if e may not have the highest influence in an individual snapshot.

In this paper, we propose two strategies – the horizontal-compression-based strategy (HCS) and the vertical-compression-based strategy (VCS) – to tackle the DIM problem. To capture the randomness of distinct influence spread under the stochastic diffusion model, we randomly generate subgraphs from snapshots of the social network via Monte Carlo simulations, and approximate the users' distinct influence spread by averaging their distinct reachability on these subgraphs. To mitigate the issue of high memory usage incurred by the generated subgraphs, our proposed strategies first compress the generated subgraphs in different ways, and then find the target users on the corresponding compressed graphs. Our main contributions are summarized as follows:

- Motivated by realistic viral marketing factors, we formalize the distinct influence maximization (DIM) problem which aims to find a fixed seed set of k target users to maximize the influence spread over distinct users in an evolving social network (Section III). The DIM problem is NP-hard and more computationally expensive than the IM problem, which can be shown to be a special case of the DIM problem.
- We propose two strategies called HCS and VCS to tackle the DIM problem (Section IV). These two strategies compress the generated subgraphs in different ways, and then identify target users using the corresponding compressed graphs. VCS is empirically shown to be superior to HCS w.r.t. running time and memory footprint as it reduces overlaps among the generated subgraphs, and can also be applied directly to the traditional IM problem.
- We provide a complete theoretical analysis to show that HCS and VCS obtain equivalent results, and the quality of the solutions is theoretically bounded (Section V).
- We conduct extensive experiments on real-world datasets, which shows that: (1) for the DIM problem, HCS and VCS significantly outperform baselines extended from the state-of-the-art methods for the IM problem in terms of memory costs, while maintaining high-quality solutions and high efficiency; (2) for the IM problem, VCS obtains accurate results while providing good trade-offs between running time and memory consumption (Section VI).

II. Related Work

A. Influence Maximization in Static Social Networks

Kempe et al. [2] proved the NP-hardness of the influence maximization problem and proposed a Monte Carlo simulation based greedy algorithm to solve the problem. Their approach iteratively selects nodes with the highest influence spread into the seed set. In each iteration, the algorithm runs a large number of simulations for each node to estimate the the influence spread. Although this algorithm produces near-optimal solutions, it is not efficient, which was the motivation in subsequent studies. Despite significant progress, solutions that are both scalable and accurate are still rare. We categorize existing approaches into four classes: simulation-based, subgraph-based, sketch-based and heuristic-based.

Simulation-based Methods. Leskovec et al. [3] adopted an early termination technique CELF to reduce the number of unnecessary estimates while achieving the same accuracy as the greedy algorithm [2]. However, CELF does not improve the worst case time complexity. Goyal et al. [4] proposed an algorithm which further reduced the number of simulations required, but in practice the new approach was not more efficient than CELF empirically [27].

Subgraph-based Methods. SGDU [5] reused a small number of subgraphs from the network that were generated with Monte Carlo simulations to compute influence spread, which greatly reduced the number of simulations needed. PMC [6] further improved the efficiency of reusing the generated subgraphs by transforming the subgraphs into acyclic graphs and introducing additional pruning techniques. However, the scalability of PMC remains limited due to the high memory costs of storing the subgraphs.

Sketch-based Methods. Borgs et al. [7] proposed a strategy to estimate influence spread by building sketches for randomly selected nodes. For a selected node r, the sketch is constructed via Monte Carlo simulations, and contains a *reverse reachable* (RR) set storing nodes which can reach r in the sketch. This strategy uses the classic greedy algorithm from the maximum coverage problem [28] to select a seed set required to cover the maximum number of RR sets. Subsequent studies such as TIM [8] and IMM [9] have been proposed to lower the number of RR sets required while still ensuring the same theoretical bound. Sketch-based strategies are susceptible to high memory costs as many RR sets may be required in order to achieve a tight theoretical bound.

Heuristic Methods. Score-estimation methods [10]–[15] such as EasyIM [12] estimate the influence of a node by exploiting the exponential decrease of influence probability w.r.t. the path length. Alternatively, given an initial ordering of nodes, a rank-refinement method such as IMRank [16] iteratively reorders the nodes based on their computed influence spread until the ranking converges. These heuristic methods can either improve efficiency or reduce memory costs. However, the solution quality is often reduced.

B. Influence Maximization in Dynamic Social Networks

In reality, social networks are constantly changing, and so the relationships among people and influence patterns are dynamic. Thus, there are recent studies focusing on how to efficiently update a seed set in dynamic social networks. Chen

et al. [19] modeled a dynamic social network as a sequence of snapshots and aimed to continuously extract seed sets for each snapshot. Ohsaka et al. [21] proposed a fully dynamic scheme which efficiently updates the seed set in real-time with every node/edge update. Wang et al. [20] modeled dynamic social networks as social streams and studied how to efficiently update the seed set over the most recent social actions.

These studies differ from ours in the following perspectives: (1) *Updated seed sets* versus *fixed seed sets*. Previous solutions continuously update the seed set while we aim to find a fixed seed set over a period of time in an evolving social network. (2) *Local maximum influence spread* versus *global maximum influence spread*. Previous solutions aimed to find a seed set which influences the maximum number of nodes at a particular instant in time – for example the most current seed set in a dynamic social network. The primary goal in previous work was to iteratively update the seed set based on the current network configuration, while we aim to find a seed set which influences the maximum number of distinct nodes across multiple snapshots of an evolving network.

III. Problem formulation & preliminaries

We represent a social network as a directed graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. We say that node u can reach node v if there exists a path from u to v in G, and we denote ***reachability*** of u as the number of nodes u can reach.

A. The Diffusion Model & Problem Formulation

In this paper, we focus on solving our proposed problem under a variation of the *Independent Cascade* (IC) model [2] – a classic and widely-adopted information diffusion model. The IC model originates from the marketing literature [29] and independently assigns each edge (u, v) with an influence probability $p_{u,v} \rightarrow [0, 1]$. The information diffusion instance unfolds in discrete steps. Given an active seed set S in time step 0, each active node u in time step $t \geq 1$ will have a single chance to activate each inactive outgoing neighbor v with a probability of $p_{u,v}$. If an outgoing neighbor v is activated in the current step t, it will become active in step $t + 1$ and then will have a single chance to activate each of the inactive outgoing neighbors in the next time step. Otherwise, the neighbor will stay inactive. The information diffusion instance terminates when no more nodes can be activated. The ***influence spread*** of a seed set S in a graph G (denoted as $\sigma_G(S)$) is the expected number of activated nodes when S is the initial active node set. Given a graph $G = (V, E)$, a positive integer k, the traditional ***influence maximization problem (IM)*** under the IC model is to find a seed set S^* of k nodes such that

$$\sigma_G(S^*) = \arg\max_{S \subseteq V \wedge |S| \leq k} \sigma_G(S)$$

In this paper, we model an evolving social network as a series of evolutionary stages, operationalized as a sequence of snapshot graphs $D = \{G^1, G^2, \ldots, G^w\}$ as in [19]. The ***distinct influence spread*** $\zeta_D\{S\}$ of a seed set S is the expected total number of distinct nodes (nodes with different IDs) which are activated by S in all of the snapshots. We simulate the diffusion process for S on each of the snapshots individually using the IC model, but nodes with the same IDs across different snapshots are only counted once. We formally define the distinct influence maximization problem as follows.

Problem 1 (Distinct Influence Maximization (DIM)). Given a sequence of snapshots $D = \{G^1, G^2, \ldots, G^w\}$ of an evolving social network, where $G^i = (V^i, E^i)$ and $1 \leq i \leq w$, a common node set $V_c = \bigcap_{i=1}^{w} V^i$ and a positive integer k, the distinct influence maximization problem aims to find a seed set S^* of k nodes such that

$$\zeta_D(S^*) = \arg\max_{S \subseteq V_c \wedge |S| \leq k} \zeta_D(S).$$

Note that since the traditional influence maximization problem is NP-hard and is a special case of the DIM problem when $w = 1$, the DIM problem is also NP-hard. We assume that an evolving social network has numerous periodic patterns, namely periodically recurring interaction patterns in networks that change over time [30]. The evolution of social networks is influenced by these periodic patterns, and patterns found in the near past (e.g. in the last day/week) can be used to guide future decisions.

B. Extension of Existing Methods to the DIM problem

To obtain the expected distinct influence spread of a seed set S on multiple graphs, we need to consider every possible case under the stochastic diffusion model. However, computing the expected distinct influence spread is #P-hard, because computing the expected influence spread on the traditional IM problem is known to be #P-hard [13] and is a special case of computing the expected distinct influence spread with a window size $w = 1$. Instead, we can approximate the expected distinct influence spread by simulating a number of (but not all) possible cases under the stochastic diffusion model.

Heuristic methods on the IM problem either estimate the influence spread of a node by using a function of the number of simple paths starting at this node [10]–[12], or reorder the nodes based on their ranking-based marginal influence spread computed by exploiting influence probabilities of edges among higher-ranked nodes and lower-ranked nodes [16]. Heuristic methods are not easily amenable to the DIM problem since they either do not use simulations, or "partially" probe a limited number of paths.

The simulation-based, subgraph-based and sketch-based methods are potential candidates for the extension since subgraphs and sketches are both constructed via simulations. Therefore, we construct our baselines using state-of-the-art methods selected from these three categories.

C. Preliminaries

In this subsection, we will describe some preliminaries to facilitate the illustration of our solutions and theoretical analysis in Section IV and Section V.

A Greedy Strategy with Theoretical Guarantees. Kempe et al. [2] proved that the IM problem is NP-hard, and proposed a greedy algorithm with an approximation ratio of $(1-1/e)$ based

Notation	Description
G^i	The i-th snapshot.
G^i_j	The j-th subgraph generated from the snapshot G^i.
$D = \{G^1, ..., G^w\}$	A sequence of snapshots with the window size w.
S	A seed set.
$\mathcal{I}_G(S)$	The set of nodes which can be reached by S in G.
$\zeta_D(S)$	The expected distinct influence spread of S in D.
G^j_{hc}	The j-th horizontally-compressed graph.
G^i_{vc}	The i-th vertically-compressed graph.
$v.\mathfrak{B}^c$	The containment bitset of node v.
$v.\mathfrak{B}^l$	The local containment bitset of node v.
$v.\mathfrak{B}^t$	The traversal bitset of node v.
$v.\mathfrak{B}^r$	The recording bitset of node v.

TABLE I: Notations.

on the monotonic and submodular properties of $\sigma(\cdot)$. Given an influence spread function $\sigma(\cdot)$ and a graph $G = (V, E)$, we say $\sigma_G(\cdot)$ is monotone if and only if $\sigma_G(S) \leq \sigma_G(S')$ for any $S \subseteq S' \subseteq V$, and submodular if and only if $\sigma_G(S \cup \{v\}) - \sigma_G(S) \geq \sigma_G(S' \cup \{v\}) - \sigma_G(S')$ for any $S \subseteq S' \subseteq V$ and $v \in V \setminus S'$. This greedy algorithm iteratively adds a node t with the maximum marginal gain $t = \arg\max_{v \in V \setminus S} \sigma_G(S \cup \{v\}) - \sigma_G(S)$, into the seed set S until S is of size k, with an approximation ratio of $1 - 1/e$ proven in Theorem 1.

Theorem 1: [28] If an influence function σ is monotone, submodular and $\sigma(\emptyset) = 0$, then for a seed set S returned by the greedy strategy, we have $\sigma(S) \geq (1 - 1/e)\sigma(S^*)$ where S^* is the optimal solution.

The Subgraph-based Strategy. To avoid a large number of Monte Carlo simulations to obtain solutions of high quality, follow-on studies [5], [6] focused on resolving the IM problem on a small number of subgraphs randomly generated via Monte Carlo simulations. A subgraph is generated by keeping each edge (u, v) with probability $p_{u,v}$ and removing each edge (u, v) with probability $1 - p_{u,v}$ from the original graph. With Theorem 2, this strategy approximates the influence spread of any seed set by estimating its average reachability on the generated subgraphs with theoretical guarantees [6].

Theorem 2: [2] Given a graph $G = (V, E)$ and a subgraph G_r randomly generated via Monte Carlo simulations, the distribution $\mathfrak{D}_G(S)$ over the sets of nodes activated by the seed set S under the IC model is same as the distribution $\mathfrak{D}_{G_r}(S)$ of the node set reachable from S in G_r.

IV. Algorithms

As a corollary of Theorem 2, the distribution over the sets of nodes activated by the seed set across different snapshots is the same as the distribution of the node set reachable from the seed set in the generated subgraphs of the independent snapshots. Thus we can approximate the distinct influence spread of any seed set by estimating its ***distinct reachability*** in the generated subgraphs of these snapshots where the same nodes across snapshots are only counted once in the reachability test.

In this section, we will optimize the subgraph-based strategy to solve the DIM problem. We observe that the generated subgraphs are memory intensive, and the efficiency of estimating reachability is sensitive to node and edge overlaps across different subgraphs in the DIM problem. Thus, we propose two strategies HCS and VCS respectively to compress these subgraphs and conduct reachability tests on the compressed graphs which eliminate node and edge overlaps.

A. The Horizontal-Compression-Based Strategy (HCS)

Suppose that we have a sequence of snapshots $D = \{G^1, G^2, \ldots, G^w\}$ with their R generated subgraphs. Let G^i_j denote the j-th subgraph generated from the snapshot G^i ($1 \leq j \leq R$, $1 \leq i \leq w$), and $\mathcal{I}_{G^i_j}(S)$ denotes the set of nodes which can be reached by the seed set S in G^i_j. Let the set of subgraphs $\{G^1_j, G^2_j, \ldots, G^w_j\}$ be the j-th ***horizontal instance***, an allowable combination of the subgraphs of each snapshot. Then the average distinct reachability of a seed set can be estimated on horizontal instances as the approximate distinct influence spread such that:

$$\zeta_D(S) = \frac{1}{R} \sum_{j=1}^{R} | \bigcup_{i=1}^{w} \mathcal{I}_{G^i_j}(S)| \tag{1}$$

To compute the distinct reachability of S in a horizontal instance, a naive way is to obtain the sets of nodes reached by S in each individual subgraph and perform union operations over these sets to generate a set of distinct users who are influenced by S. However, it is inefficient and incurs high memory overhead when storing these subgraphs. In addition, the overlapping nodes and edges across different snapshots may be visited multiple times during the estimations, which further decreases the efficiency. To mitigate these issues, we will introduce the Horizontal Compression technique which compresses subgraphs of each horizontal instance into one single graph, followed by an algorithm which performs distinct reachability tests directly on the compressed graphs.

1) Horizontal Compression: In the compressed graph, each node (edge) is assigned with a ***containment bitset*** $\mathfrak{B}^c$ storing information about which subgraphs contain this node (edge). Suppose we have a horizontal instance $\{G^1_j, G^2_j, \ldots, G^w_j\}$ ($1 \leq j \leq R$), an empty ***horizontally compressed graph*** G_{hc} and an empty containment bitset $\mathfrak{B}^c$ of size w where $\mathfrak{B}^c[i] = 0$ ($1 \leq i \leq w$). We construct G_{hc} by scanning each subgraph iteratively. When we scan through G^i_j, if an edge e in G^i_j does not exist in G_{hc}, we add e into G_{hc}, assign e with $\mathfrak{B}^c$ and set $e.\mathfrak{B}^c[i] = 1$. Otherwise, we update the existing $e.\mathfrak{B}^c[i]$ in G_{hc} to 1. Since the aforementioned process of generating subgraphs only removes edges, every subgraph contains all nodes in G regardless of connectivity. Therefore, we assign a containment bitset for every node v in G_{hc} such that all bits in this containment bitset are set to 1. Eventually, G_{hc} contains all nodes and edges appearing in the subgraphs of the same horizontal instance and the overlaps are eliminated with the assigned containment bitsets. For a bitset $e.\mathfrak{B}^c$ ($v.\mathfrak{B}^c$), if $e.\mathfrak{B}^c[i] = 1$ ($v.\mathfrak{B}^c[i] = 1$) ($1 \leq i \leq w$), it indicates that the edge e (node v) exists in G^i_j. Thus, the compressed graph G_{hc} stores the information of nodes and edges in these subgraphs.

Figure 2 shows subgraphs G^1_1, G^2_1, G^3_1 of the same horizontal instance and their compressed graph G_{hc}. Note that all

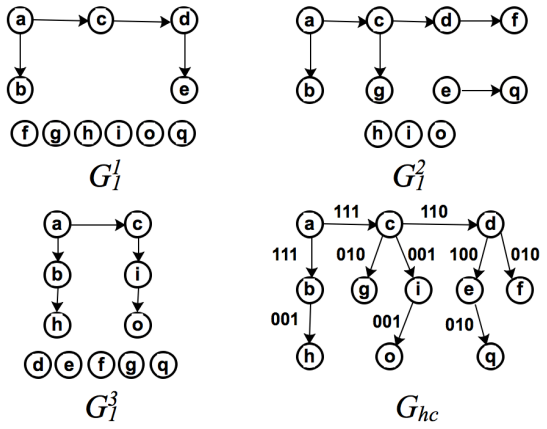


Fig. 2: An example of three subgraphs and their horizontally compressed graph.

Algorithm 1: The Horizontal-Compression-Based Strategy (HCS)

Input : Snapshots $G^1 = (V^1, E^1), ..., G^w = (V^w, E^w)$, k and R (# of subgraphs).
Output : The seed set S.

```
S, D_hc = ∅;
for j = 1 to R do
    G^j_hc ← (V^j_hc = ∅, E^j_hc = ∅);
    for i = 1 to w do
        generate subgraph G^i_j to update G^j_hc ;
    Add G^j_hc into D_hc.
while |S| < k do
    t ← arg max_{v∈V_c} HCA(D_hc, v);
    S ← S ∪ {t};
    Update(D_hc, t);
return S;
```

disconnected nodes are placed together. Each node in G_{hc} has a containment bitset $\{111\}$ not shown in Figure 2. For each edge e in G_{hc}, if e exists in G_1^i $(1 \leq i \leq 3)$, then $e.\mathfrak{B}^c[i] = 1$. For example, the containment bitset $(c,d).\mathfrak{B}^c$ is $\{110\}$ since the edge (c,d) exists in G_1^1 and G_1^2.

Space Complexity Analysis. Regardless of influence probabilities assigned to edges in snapshots $\{G^1, G^2, \ldots, G^w\}$ where $G^i = (V^i, E^i)$ $(1 \leq i \leq w)$, we can construct a ***merged snapshot*** $G^M = (V^M, E^M)$ such that $V^i \subseteq V^M$ and $E^i \subseteq E^M$. Recall the process of constructing subgraphs, a subgraph contains all nodes and edges of the corresponding snapshot in the worst case. Thus, a compressed graph of a horizontal instance can have at most $O(|V^M| + |E^M|)$ nodes and edges. Suppose that each containment bitset of size w uses $\mathcal{C}_w$ space. Then a horizontally compressed graph consumes $O((1 + \mathcal{C}_w)(|V^M| + |E^M|))$ space in total.

2) Algorithms: Algorithm 1 describes the strategy of HCS which conducts estimations on horizontally compressed graphs. It consists of four main steps: (1) construct the compressed graphs; (2) estimate the average distinct reachability of each node in the compressed graphs using HCA (Algorithm 2); (3) iteratively add a common node with the maximum marginal gain w.r.t. the average distinct reachability into the seed set; (4) update the compressed graphs each time a seed is selected.

In our proposed algorithm HCA (Algorithm 2), the estimates on the horizontally compressed graphs aim to achieve the effect of simultaneously performing breadth-first search (BFS) on the subgraphs of the same horizontal instance. First, let us introduce two data structures used in the compressed graph G_{hc}^j constructed from the j-th horizontal instance:

- ***Traversal bitset*** – Each node u in the queue to be processed is assigned with a *traversal bitset* $\mathfrak{B}^t$. The traversal bitset of u describes which subgraphs can continue traversals from u. For example, if $u.\mathfrak{B}^t[i] = 1$, the subgraph G_j^i can continue visiting outgoing neighbors of u. No traversal will be allowed to continue from node u if and only if all bits in $u.\mathfrak{B}^t$ are 0.
- ***Local containment bitset*** – Each reached node u is assigned with a *local containment bitset* $\mathfrak{B}^l$ initialized as $u.\mathfrak{B}^c$. The local containment bitset $\mathfrak{B}^l$ of u describes which subgraphs contain u but have not visited u. Note that if G_j^i successfully visits u, we need to set $u.\mathfrak{B}^l[i] = 0$, which makes sure that G_j^i cannot pass u again.

Note that traversal bitsets and local containment bitsets are temporary data structures used for conducting estimations in each individual compressed graph.

Next we describe how to assign the traversal bitset for each node in the queue to be processed and update the local containment bitset of each visited node in G_{hc}^j. We say that node u can traverse to an outgoing neighbor node w if and only if the result of bitwise AND operations among the $u.\mathfrak{B}^t$, $(u,w).\mathfrak{B}^c$ in G_{hc}^j and $w.\mathfrak{B}^l$, is not 0. This is true because we can proceed with the traversal if and only there exists at least one subgraph satisfying the following ***three traversal rules***:

1) The value of the corresponding bit in $\mathfrak{B}^t$ of u is 1 $(u.\mathfrak{B}^t[i] = 1)$, indicating that G_j^i can proceed the traversal.
2) The value $(u,w).\mathfrak{B}^c[i] = 1$, indicating that G_j^i contains edge (u,w).
3) The value $w.\mathfrak{B}^l[i] = 1$, indicating that G_j^i contains w and has not visited w yet.

The result b of the bitwise AND operations stores the information about which subgraphs successfully traverse from u to w, and it is also the updated traversal bitset assigned to node w. If b is not 0, we update the local containment bitset $w.\mathfrak{B}^l$ with b by changing the corresponding bits of successful subgraphs to 0 $(w.\mathfrak{B}^l \oplus b)$. Afterwards, we push u and the updated traversal bitset b onto queues.

We use the symbol $u \overset{G_{hc}^j}{\rightsquigarrow} w$ to denote that u can reach w in G_{hc}^j. Given a sequence of snapshots $D = \{G^1, G^2, \ldots, G^w\}$ and their horizontally compressed graphs $G_{hc}^1, \ldots, G_{hc}^R$ where $G_{hc}^j = \{V_{hc}^j, E_{hc}^j\}$ $(1 \leq j \leq R)$, HCA approximates the distinct influence spread of a seed set S as follows:

Algorithm 2: HORIZONTAL-COMPRESSION-BASED ALGORITHM (HCA)

Input : $G^1_{hc} = \{V^1_{hc}, E^1_{hc}\}, ..., G^R_{hc} = \{V^R_{hc}, E^R_{hc}\}$, and node v.
Output : The average distinct reachability of v.

1 $d \leftarrow 0$;
2 **for** $j = 1$ *to* R **do**
3 **if** $v.\mathfrak{B}^c$ *in* $G^j_{hc} \neq 0$ **then**
4 $d \leftarrow d + 1$;
5 $Q_1 \leftarrow$ a queue initialized with node v;
6 $Q_2 \leftarrow$ a queue initialized with $v.\mathfrak{B}^c$ in G^j_{hc};
7 **while** Q_1 *is not empty* **do**
8 $u \leftarrow$ dequeue from Q_1;
9 $\mathfrak{B}^t \leftarrow$ dequeue from Q_2;
10 **foreach** *edge* $(u, x) \in E^j_{hc}$ **do**
11 **if** x *is not visited in* G^j_{hc} **then**
12 $x.\mathfrak{B}^l \leftarrow x.\mathfrak{B}^c$ in G^j_{hc};
13 $b \leftarrow \mathfrak{B}^t$ & $(u, x).\mathfrak{B}^c$ in G^j_{hc} & $x.\mathfrak{B}^l$;
14 **if** $b \neq 0$ **then**
15 **if** $x.\mathfrak{B}^l = x.\mathfrak{B}^c$ *in* G^j_{hc} **then**
16 $d \leftarrow d + 1$;
17 $x.\mathfrak{B}^l \leftarrow x.\mathfrak{B}^l \oplus b$;
18 $Q_1 \leftarrow$ enqueue the node x;
19 $Q_2 \leftarrow$ enqueue the bitset b;
20 **return** d/R;

$$\zeta_D(S) = \frac{1}{R} \sum_{j=1}^{R} \sum_{cond} 1 \tag{2}$$

where *cond* is the condition that $x \in V^j_{hc} \wedge \exists t \in S,\ t \overset{G^j_{hc}}{\rightsquigarrow} x$.

Algorithm 2 shows our proposed method HCA. In each iteration, a BFS on a compressed graph is performed. If the bitset of the starting node v is not 0, we do the following operations: (1) Increment the distinct reachability d since v can reach itself; (2) Initialize two queues: Q_1 that stores the nodes to be visited and is initialized with the starting node v; Q_2 that stores traversal bitsets of nodes to be visited and is initialized with the containment bitset of the starting node v; (3) Process each node in Q_1 until Q_1 is empty. In each step, we dequeue one node u from Q_1 along with its corresponding traversal bitset from Q_2. We check whether we can traverse from u to its outgoing neighbors based on whether the value of b (the updated traversal bitset) is not 0 (Lines 13-14). If not, we update the local containment bitset of the neighbor, and enqueue this neighbor with its traversal bitset onto the queues, and increment the distinct reachability d if the neighbor has not been visited (Lines 14-19); (4) When the BFS finishes, we return the average distinct reachability (Lines 20).

To update the compressed graphs, we conduct a similar traversal from the selected seed t as HCA but update containment bitsets $\mathfrak{B}^c$ of the visited nodes instead.

Correctness Analysis. To prove the correctness of HCA, we need to show that the result returned by HCA (Formula 2) is equivalent to Formula 1.

Lemma 3: Given a horizontal instance $\{G^1_j, G^2_j, \ldots, G^w_j\}$ and the compressed graph G^j_{hc}, if node u can reach node x in G^i_j $(1 \leq i \leq w)$, then node u can reach node x in G^j_{hc}.

Proof. Recall the function of the traversal bitset. If we can show that the updated traversal bitset $\mathfrak{B}^t$ has at least one bit of 1 when we traverse from u to x along a path, we can then prove that u can reach x in G^j_{hc}. For each edge (a, b) to be traversed, the traversal bitset $\mathfrak{B}^t$ is updated by the bitwise AND operations among the current $\mathfrak{B}^c$ carried by a, the containment bitset $(a, b).\mathfrak{B}^c$ in G^j_{hc} and the local containment bitset $\mathfrak{B}^l$ of b initialized as $b.\mathfrak{B}^c$ (refer to the three traversal rules). There must exist a path p in G^j_{hc} such that $z.\mathfrak{B}^c[i] = 1$ and $e.\mathfrak{B}^c[i] = 1$ for every node z and edge e along p, since node u can reach node x in G^i_j. Considering that $\mathfrak{B}^t$ is initialized as $u.\mathfrak{B}^c$ in G^j_{hc}, the i-th bit of the updated $\mathfrak{B}^t$ must always be 1 during the traversal from u to x along p for the first time. Thus, u can reach x in G^j_{hc}.

Theorem 4: Given a sequence of snapshots $D = \{G^1, G^2, \ldots, G^w\}$ each of which has R generated subgraphs, the horizontally-compressed graphs $G^1_{hc}, \ldots, G^R_{hc}$, and a seed set S, the result returned by the HCA is the average distinct reachability of S on these compressed graphs such that

$$\zeta_D(S) = \frac{1}{R} \sum_{j=1}^{R} |\bigcup_{i=1}^{w} \mathcal{I}_{G^i_j}(S)| = \frac{1}{R} \sum_{j=1}^{R} \sum_{cond} 1 \tag{3}$$

where *cond* is the condition that $x \in V^j_{hc} \wedge \exists t \in S,\ t \overset{G^j_{hc}}{\rightsquigarrow} x$.

Proof. As a direct corollary of Lemma 3, in the horizontally-compressed graph G^j_{hc} $(1 \leq j \leq R)$, the seed set S can reach all the nodes reached by S in any individual subgraph of the j-th horizontal instance. Since we only count each reached node once in G^j_{hc} and estimations on all compressed graphs are independent, Formula 3 is correct.

Time Complexity Analysis. As we mentioned before, given snapshots $\{G^1, \ldots, G^w\}$ and their merged snapshot $G^M = (V^M, E^M)$, a compressed graph can have at most $O(|V^M|)$ nodes and $O|E^M|$ edges. The method HCA aims to estimate the reachability of a node on each subgraph simultaneously. Each compressed graph G_{hc} is constructed from w subgraphs and the reachability test for each individual subgraph compressed in G_{hc} traverses at most $O(|E^M|)$ edges. In addition, each traversal involves a fixed number bitwise operations each of which takes $O(\mathcal{B})$ time. Thus, Algorithm 2 takes at most $O(R\mathcal{B}w|E^M|)$ time for R compressed graphs. Let m_{avg} be the average number of edges of the sequence of snapshots. Algorithm 1 takes $O(Rwm_{avg})$ time to construct R compressed graphs, calls the method HCA $O(k|V^M|)$ times and takes $O(Rk\mathcal{B}w|E^M|)$ time in total to update the compressed graphs. Hence, the total time complexity of the HCS strategy is $O(R\mathcal{B}kw|E^M||V^M| + Rwm_{avg})$. Note that a distinct reachability estimation on a compressed graph costs much less than $O(\mathcal{B}w|E^M|)$ empirically since there are often overlapping

subgraphs, and we can perform an edge traversal on multiple subgraphs simultaneously in $O(\mathcal{B})$ time.

B. The Vertical-Compression-Based Strategy (VCS)

Since the number of node and edge overlaps across subgraphs generated from the same snapshot is more likely to be greater than the one across subgraphs generated from different snapshots, the memory costs can be further reduced if subgraphs generated from the same snapshot are compressed instead. Suppose we have a sequence of snapshots $D = \{G^1, G^2, \ldots, G^w\}$ and their generated subgraphs G_j^i ($1 \leq j \leq R$, $1 \leq i \leq w$). Let the set of subgraphs $\{G_1^i, G_2^i, \ldots, G_R^i\}$ be the i-th ***vertical instance***. We can approximate the distinct influence spread of S based on vertical instances as follows:

$$\zeta_D(S) = \frac{1}{R}\sum_{i=1}^{w}\sum_{j=1}^{R}|\mathcal{I}_{G_j^i}(S) - \bigcup_{l=1}^{i-1}\mathcal{I}_{G_j^l}(S)| \tag{4}$$

Note that Formula 4 and Formula 1 are equivalent.

1) Vertical Compression: We adopt a similar compression procedure as Horizontal Compression but Vertical Compression compresses subgraphs of the same vertical instance into a ***vertically-compressed graph*** G_{vc}, and thus the size of the containment bitset assigned to each node (edge) is same as the size of a vertical instance.

Space Complexity Analysis. Suppose we have a sequence of snapshots $\{G^1, G^2, \ldots, G^w\}$ where $G^i = (V^i, E^i)$ ($1 \leq i \leq w$), and the resulting R generated subgraphs. Since a vertically-compressed graph is still a subgraph of the corresponding snapshot, all vertically-compressed graphs can have at most $O(\sum_{i=1}^{w}(|V^i| + |E^i|))$ nodes and edges. If each containment bitset of size R consumes $O(\mathcal{C}_R)$ space, all compressed graphs consume $O((1+\mathcal{C}_R)(\sum_{i=1}^{w}(|V^i| + |E^i|))$ space. Let $m_{avg} = \frac{1}{w}\sum_{i=1}^{w}|E^i|$ and $n_{avg} = \frac{1}{w}\sum_{i=1}^{w}|V^i|$, then one compressed graph uses $O((1+\mathcal{C}_R)(n_{avg} + m_{avg}))$ space.

2) Algorithms: The VCS strategy follows the four steps which are conceptually similar to Strategy HCS, but VCS constructs vertically-compressed graphs in the first step. We omit the pseudocode due to space limitations. Algorithm 3 shows the main idea of our proposed algorithm VCA which computes average distinct reachability based on vertically-compressed graphs. VCA differs from HCA w.r.t. the following four points: (1) results are obtained on vertically-compressed graphs with reduced memory costs; (2) each visited node x is assigned an extra data structure called a ***recording bitset*** $\mathfrak{B}^r$. We say v can reach x in a horizontal instance if v can reach x in at least one subgraph of this horizontal instance. The recording bitset of x records in which horizontal instances v have visited x and is updated (line 17 and 19) by the result b of bitwise operations mentioned previously (line 13). The result b records which subgraphs of a vertical instance have successfully visited x and each successful subgraph also belongs to a specific horizontal instance. We use $Count(x.\mathfrak{B}^r)$ to denote the process of computing the number of such horizontal instances (popcount); (3) an additional queue Q_3 is used to store the visited nodes. After the main loops, we get the result by averaging the sum of how many horizontal instances have visited each node in Q_3 (line 23 to 25).

Algorithm 3: VERTICAL-COMPRESSION-BASED ALGORITHM (VCA)

Input : $G_{vc}^1 = \{V_{vc}^1, E_{vc}^1\}, ..., G_{vc}^w = \{V_{vc}^w, E_{vc}^w\}$, and node v.
Output : The average distinct reachability of v.

```
1  v.B^r, d ← 0;
2  Q_3 ← a queue initialized with node v;
3  for i = 1 to w do
4      v.B^r ← v.B^r | v.B^c in G_vc^i;
5      Q_1 ← a queue initialized with node v;
6      Q_2 ← a queue initialized with bitset v.B^c in G_vc^i;
7      while Q_1 is not empty do
8          u ← dequeue from Q_1;
9          B^t ← dequeue from Q_2;
10         foreach edge (u, x) ∈ E_vc^i do
11             if x is not visited by G_vc^i then
12                 x.B^l ← x.B^c in G_vc^i;
13             b ← B^t & (u, x).B^c in G_vc^i & x.B^l;
14             if b ≠ 0 then
15                 if x is not in Q_3 then
16                     Q_3 ← enqueue the node x;
17                     x.B^r ← b;
18                 else
19                     x.B^r ← x.B^r | b;
20                 w.B^l ← w.B^l ⊕ b;
21                 Q_1 ← enqueue the node x;
22                 Q_2 ← enqueue the bitset b;
23 foreach node x in Q_3 do
24     d ← d + Count(x.B^r);
25 return d/R;
```

Therefore, given a sequence of snapshots $D = \{G^1, \ldots, G^w\}$, their vertically-compressed graphs $G_{vc}^1, \ldots, G_{vc}^w$ constructed from w vertical instances, a seed set S and a queue Q storing nodes visited by S in vertically-compressed graphs, VCA approximates the distinct influence spread of a seed set S as follows:

$$\zeta_D(S) = \frac{1}{R}\sum_{x \in Q} Count(x.\mathfrak{B}^r) \tag{5}$$

To update compressed graphs, we conduct a similar traversal from the selected seed t as VCA but update containment bitsets of visited nodes instead of their local containment bitsets.

Correctness Analysis. Since Formula 1 and Formula 4 are equivalent, we can prove that VCA obtains the same result as HCA if we can show that Formula 5 is equivalent to Formula 4.

Lemma 5: Given a vertical instance $\{G_1^i, G_2^i, \ldots, G_R^i\}$ and the compressed graph G_{vc}^i, if node u can reach node x in G_j^i ($1 \leq i \leq R$), then node u can reach node x in G_{vc}^i.

Proof. The proof is similar to Lemma 3. If node u can reach node x in G_j^i, there must exist a path p in G_{vc}^i such

that $z.\mathfrak{B}^c[i] = 1$ and $e.\mathfrak{B}^c[i] = 1$ for every node z and edge e along p. The updated traversal bitset $\mathfrak{B}^t$ has at least one bit of 1 when we traverse from u to x along p, since $\mathfrak{B}^t$ is initialized as $u.\mathfrak{B}^c$ in G^i_{vc} and the i-th bit of the updated $\mathfrak{B}^t$ must always be 1 during the traversal from u to x for the first time. Thus, u can reach x in G^i_{vc}.

Theorem 6: Given a sequence snapshots $D = \{G^1, G^2, \ldots, G^w\}$ together with their R generated subgraphs, a seed set S, and a queue Q storing nodes visited by S in the vertically-compressed graphs, the result returned by VCA is the average distinct reachability of S such that

$$\begin{aligned}\zeta_D(S) &= \frac{1}{R}\sum_{i=1}^{w}\sum_{j=1}^{R}|\mathcal{I}_{G^i_j}(S) - \bigcup_{l=1}^{i-1}\mathcal{I}_{G^l_j}(S)| \\ &= \frac{1}{R}\sum_{x\in Q} Count(x.\mathfrak{B}^r)\end{aligned} \tag{6}$$

Proof. For a horizontal instance $\{G^1_j, G^2_j, \ldots, G^w_j\}$ $(1 \leq j \leq R)$, its subgraphs belong to w different vertical instances and thus are compressed into w different vertically-compressed graphs. Based on Lemma 5, after we conduct reachability tests of S on all vertically-compressed graphs, the queue Q stores all the nodes which can be reached by S in any subgraph of any horizontal instance. Recall that we introduce a recording bitset $\mathfrak{B}^r$ for each node x reached by S in any vertically-compressed graph. If the i-th bit of $\mathfrak{B}^r$ is 1, it indicates that some subgraphs of the i-th horizontal instance have reached x in the corresponding vertically-compressed graphs. The recording bitset $\mathfrak{B}^r$ only gets updated when some subgraphs of the current vertically-compressed graphs reach nodes which cannot be reached by the subgraphs of the same horizontal instances but belonging to previous vertically-compressed graphs (line 19). By exploiting the recording bits of nodes stored in Q, we can know how many distinct nodes are reached in total, and which nodes S can reach in each horizontal instance (line 24). Thus, Formula 6 is correct.

Time Complexity Analysis. Suppose we have a sequence of snapshots $G^1 = (V^1, E^1), \ldots, G^w = (V^w, E^w)$ each of which has R generated subgraphs and their merged snapshot $G^M = (V^M, E^M)$. Let $m_{avg} = \frac{1}{w}\sum_{i=1}^{w}|E^i|$ and $n_{avg} = \frac{1}{w}\sum_{i=1}^{w}|V^i|$. Each vertically-compressed graph G_{vc} is constructed from R subgraphs and the reachability test for each individual subgraph compressed in G_{vc} traverses at most $O(m_{avg})$ edges. Each traversal involves a fixed number of bitwise operations each of which takes $O(\mathcal{B})$ time. Thus, Algorithm 3 takes $O(\mathcal{B}Rwm_{avg})$ time. Strategy VCS runs in $O(\mathcal{B}Rkwm_{avg}|V^M| + Rwm_{avg})$ time, since it takes $O(Rwm_{avg})$ times to construct the compressed graphs, calls Algorithm 3 $O(k|V^M|)$ times and takes $O(\mathcal{B}Rkwm_{avg})$ time in total to update the compressed graphs.

Space complexity comparison between HCS and VCS. Given w snapshots and their R generated subgraphs, HCS consumes $O((1+\mathcal{C}_w)(|V^M|+|E^M|)R)$ space and VCS consumes $O((1+\mathcal{C}_R)(n_{avg}+m_{avg})w)$ space. HCS is more sensitive to R than w as $O(R)$ grows faster than $O(\mathcal{C}_w)$ in practice. Similarly, VCS is more sensitive to w than R. If we assume that $O(|V^M|+|E^M|)$ and $O(n_{avg}+m_{avg})$ are similar, then HCS will be smaller than VCS if $w \gg R$. If high quality results are required and R is large, VCS is the superior choice as $w > R$ is rare in many practical situations. Moreover, $O(|V^M|+|E^M|)$ is often notably larger than $O(n_{avg}+m_{avg})$ empirically. So cases where HCS clearly outperforms VCS are uncommon.

Speed up techniques. To reduce the number of distinct reachability estimations, we adopt the early termination technique proposed by Leskovec et al. [3] and leverage the submodularity property of the greedy strategy.

V. Theoretical properties

According to Theorem 1, the greedy strategy in the traditional IM problem which iteratively selects a seed with the maximum marginal gain returns results with theoretical guarantees since the influence function $\sigma(\cdot)$ is monotone and submodular. In this section, we will first prove the submodularity and mononicity of the influence function $\zeta(\cdot)$ such that the greedy strategy in the DIM problem is also theoretically bounded. Next, we will prove that average distinct reachability on the generated subgraphs of snapshots can approximate the distinct influence spread. Finally, we will introduce a speed-up technique used to improve the efficiency of our strategy.

A. Submodularity and Monotonicity

Theorem 7: Given a sequence of snapshot graphs $D = \{G^1, \ldots, G^w\}$ where $G^i = (V^i, E^i)$ $(1 \leq i \leq w)$, and there is a common node set $V_c = \bigcap_{i=1}^{w} V^i$, the influence function $\zeta_D(\cdot)$ is submodular such that:

$$\zeta_D(S \cup \{v\}) - \zeta_D(S) \geq \zeta_D(S' \cup \{v\}) - \zeta_D(S')$$

for any $S \subseteq S' \subseteq V_c$ and $v \in (V_c - S)$.

Proof. As a corollary of Theorem 2, the distributions of the set of nodes activated by a seed set S in the snapshots is same as the distributions of the set of nodes reached by S in the subgraphs randomly generated via Monte Carlo simulations from these independent snapshots. Let a horizontal instance be a sample point X which is one possible combination of subgraphs $G^1_X, \ldots, G^w_X$ randomly generated from these snapshots. Then the set of distinct users activated by S in X is $\bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(S)$. Let $\zeta_D(S, X) = \bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(S)$ denote the distinct influence spread of the seed set S under the sample point X. Then the expression $\zeta_D(S \cup \{v\}, X) - \zeta_D(S, X)$ is the size of the set of distinct nodes which can be reached by v but not by S. Thus, we have

$$\zeta_D(S \cup \{v\}, X) - \zeta_D(S, X) = |\bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(\{v\}) - \bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(S)|$$

Similarly, for the seed set S', we have

$$\zeta_D(S' \cup \{v\}, X) - \zeta_D(S', X) = |\bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(\{v\}) - \bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(S')|$$

Since $S \subseteq S' \subseteq V$, then $\bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(S) \subseteq \bigcup_{i=1}^{w}\mathcal{I}_{G^i_X}(S')$. Thus, we have

$$\zeta_D(S \cup \{v\}, X) - \zeta_D(S, X) \geq \zeta_D(S' \cup \{v\}, X) - \zeta_D(S', X)$$

Considering the entire space of all sample points, the set of nodes activated by the seed set S is computed by the weighted average over all outcomes under different sample points, which can be expressed as follow:

$$\zeta_D(S) = \sum_{all\ samples\ X} Prob[X] \cdot \zeta_D(S, X)$$

Since a non-negative linear combination of submodular functions is also submodular, the function $\zeta_D(\cdot)$ is submodular.

Theorem 8: Given a sequence of snapshot graphs $D = \{G^1, \ldots, G^w\}$ where $G^i = (V^i, E^i)$ $(1 \leq i \leq w)$, and the common node set $V_c = \bigcap_{i=1}^{w} V^i$, the influence function $\zeta_D(\cdot)$ is monotone such that:

$$\zeta_D(S) \leq \zeta_D(S')$$

for any $S \subseteq S' \subseteq V_c$.

Proof. For a sample point X, we have $\zeta_D(S, X) = \bigcup_{i=1}^{w} \mathcal{I}_{G^i_X}(S)$, $\zeta_D(S', X) = \bigcup_{i=1}^{w} \mathcal{I}_{G^i_X}(S')$. Since $\bigcup_{i=1}^{w} \mathcal{I}_{G^i_X}(S) \subseteq \bigcup_{i=1}^{w} \mathcal{I}_{G^i_X}(S')$, we have $\zeta_D(S, X) \leq \zeta_D(S', X)$.

Considering the entire space of sample points, a non-negative linear combination of monotone functions is also monotone. Hence, the influence function $\zeta_D(\cdot)$ is monotone.

B. Theoretical Guarantees

In this section, we will prove the theoretical guarantees of our strategies which use different but equivalent ways to estimate the average distinct reachability as the approximate distinct influence spread. First, let us introduce some key definitions. Suppose S_i is the seed set chosen after the i-th iteration $(1 \leq i \leq k)$. Let F_i be the family of node sets for which HCS or VCS estimates the average distinct reachability in the i-th iteration such that $F_{i+1} = \{S_i \cup \{v\} | v \in \bigcap_{i=1}^{w} V^i - S_i\}$, and $F = \bigcup_{i=1}^{w} F_i$.

Lemma 9: [31] Let $Y_1, \ldots, Y_n$ be independent random variables in [0,1] and $\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$. We have $Prob[|\overline{Y} - Y| > t] \leq 2e^{-2nt^2}$.

Theorem 10: Suppose we have: (1) a sequence of snapshots $D = \{G^1, \ldots, G^w\}$ where $G^i = (V^i, E^i)$ $(1 \leq i \leq w)$; (2) a merged snapshot $G^M = (V^M, E^M)$; (3) a family of node sets F; and (4) $R = O(\frac{1}{\epsilon^2}(\log|F| + \log\frac{1}{\delta})$ sample points (horizontal instances) $X_1, \ldots, X_R$.

We have $\overline{\zeta}(S) = \zeta_D(S) \pm \epsilon|V^M|$ with at least $1 - \delta$ probability for every $S \in F$, where $\overline{\zeta}(S) = \frac{1}{R}\sum_{i=1}^{R} \zeta_D(S, X_i)$.

Proof. Given a merged snapshot $G^M = (V^M, E^M)$, we have $0 < \frac{1}{|V^M|} \leq \frac{\zeta_D(S,X_i)}{|V^M|} \leq 1$. Based on Lemma 9, we have $Prob[|\overline{\zeta}(S) - \zeta_D(S)| > \epsilon|V^M|] \leq 2e^{-2\epsilon^2 R}$ for a fixed seed set S. Thus, for every $S \in F$, we have

$$Prob[|\overline{\zeta}(S) - \zeta_D(S)| \leq \epsilon|V^M|] \geq 1 - 2e^{-2\epsilon^2 R}|F|$$

with the union bound [31] over all sets in F. Thus, we obtain the bound with $R = O(\frac{1}{\epsilon^2}(\log|F| + \log\frac{1}{\delta}))$. Given the node set $V_c = \bigcap_{i=1}^{w} V^i$, we can equivalently set $R = O(\frac{1}{\epsilon^2}(\log k + \log|V_c| + \log\frac{1}{\delta}))$ to obtain the bound since $|F| \leq k|V_c|$.

VI. Experiments

In this section, we conduct experiments for both the IM problem and our proposed DIM problem where we construct baselines by extending state-of-the-art solutions on the IM problem, and show the competitiveness of our approaches.

A. Experimental Setup

Datasets We conduct experiments on five real-world datasets *NetHEPT*, *DBLP* [2], *Hyves*, *Flixster* and *LiveJournal* [3]. Table II summarizes the characteristics of each collection.

Dataset	$\|V\|$	$\|E\|$
NetHEPT	15K	59K
DBLP	654K	2M
Hyves	1.4M	3M
Flixster	2.5M	8M
LiveJournal	5.2M	49M

TABLE II: Basic information of 5 real-world networks tested.

Environments. We conduct experiments on a Linux server with Intel Xeon E5 (2.60 GHz) CPUs and 512 GB RAM. All algorithms are implemented in C++, and ran in a single thread.
Probabilistic Settings. We adopt two classical models – the *trivalency model* which randomly assigns each edge with a probability uniformly chosen from $\{0.1, 0.01, 0.001\}$ [13], and the *constant model* which assigns each edge with a constant probability. Following previous work, we set the value to 0.1 or 0.01 [2], [12], [13], [32]. The *trivalency model* is used with all of the test collections, and performance characteristics of the algorithms are also shown for the *constant model* on the largest dataset (*LiveJournal*).

Computation of Expected Spread. We adopt the comparison methodologies for the IM problem used in previous work to compute expected influence spread by performing 10K MC simulations. The same technique is used to compute expected distinct influence spread for the DIM problem.

B. Experimental Results for the IM problem

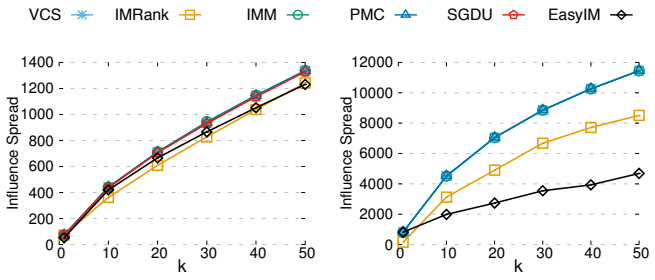
For the IM problem, we only show experimental results on *DBLP* and *LiveJournal* due to space limitations. We see similar trends in the other test collections. Additionally, only results for *LiveJournal* using the constant model (0.01) can be shown, and were the most interesting for this model. Note that we do not include SGDU in experiments on *LiveJournal* as it was unable to process a collection of this size due to limitations in the reference implementation [5].

Algorithms & Parameter Settings. We compare our strategy VCS with five state-of-the-art methods: two subgraph-based methods SGDU [5] and PMC [6]; one sketch-based method IMM [9]; two heuristic methods EasyIM [12] and IMRank [16]. Note that we do not include simulation-based methods as they are not competitive on large collections. We set all parameters (shown in Table III) following the configurations prescribed in a recent comprehensive benchmark for the IM problem [27].

[2] http://research.microsoft.com/en-us/people/weic/.
[3] http://konect.cc.

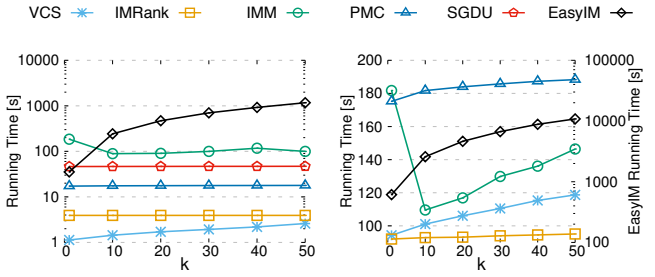
Algorithm	Parameter	
	Description	Value
VCS (our method)	#Subgraphs	200
SGDU [5]	#Subgraphs	200
PMC [6]	#Subgraphs	200
IMM [9]	The sampling error rate	0.05
EasyIM [27]	The maximum length of influence paths	3
IMRank [16]	The maximum length of influence paths	1

TABLE III: Parameter settings for the IM algorithms.



(a) DBLP (b) LiveJournal

Fig. 3: Comparison of influence spreads.



(a) DBLP (b) LiveJournal

Fig. 4: Comparison of running time.

Influence Spread. Figure 3 compares the influence spread achieved by all methods on *DBLP* and *LiveJournal* ($1 \leq k \leq 50$). The approximation methods VCS, PMC, SGDU and IMM produce almost identical results in all scenarios, with a difference in influence spread between each pair of methods of less than 0.001%. However, IMRank and EasyIM return results of measurably lower quality than the other approximation methods. Specifically, the influence spread achieved by EasyIM on LiveJournal is 40% worse than VCS, PMC and IMM, and is 55% worse than IMRank.

Running Time. Figure 4 shows the running time of each algorithm ($1 \leq k \leq 50$). Our proposed strategy VCS consistently outperforms all baselines on *DBLP* but is outperformed by IMRank on *LiveJournal*. The reason for this efficiency decay is that during a reachability estimation, a node can be revisited if its local containment bitset contains at least one bit of 1, and the likelihood increases with graph size. It is worth noting that the performance of IMM is not directly correlated with k or the size of the dataset and IMM outperforms PMC on *LiveJournal*. The performance characteristics of IMM is mainly determined by θ, the number of generated RR sets, where $\theta = \frac{\lambda}{LB}$, λ is a function of k, and LB is a computed lower bound of the optimal solution. Both λ and LB increase with k in IMM, and

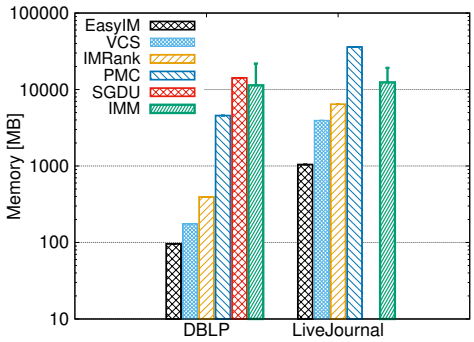


Fig. 5: Comparison of memory consumption.

Algorithm	Parameter	
	Description	Value
HCS (our method)	#Subgraphs	200
VCS (our method)	#Subgraphs	200
IMM [9]	The sampling error rate	0.05
PMC [6]	#Subgraphs	200
CELF [3]	#Simulations	10000

TABLE IV: Parameter settings for the DIM algorithms.

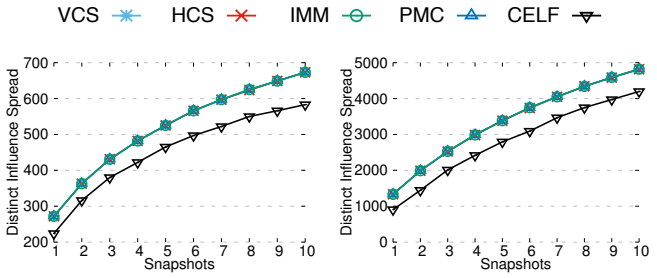
the value of θ can increase or decrease as k increased on the same dataset [9]. In addition, the value of θ may be higher on a smaller graph for a given k.

Memory Consumption. Figure 5 compares memory consumption of each algorithm. Note that only the memory consumption of IMM changes with k (shown as an error bar in Figure 5). It is worth noting that all algorithms except IMM require significantly more memory for *LiveJournal* than *DBLP*, while IMM uses a similar amount of memory on both datasets. As mentioned previously, IMM is highly sensitive to θ which can be influenced by many factors in addition to the graph size. Please refer to [9] for a more detailed analysis of θ. Among all approximation methods, VCA requires the least amount of memory, and is competitive with the heuristic methods EasyIM and IMRank. Specifically, VCS uses around three orders of magnitude less memory than the approximation methods PMC, SGDU and IMM, and uses at least 50% less memory than IMRank.

C. Experiment Results for the DIM problem

Algorithms & Parameter Settings. We compare our strategies HCS and VCS with three baselines extended from the state-of-the-art simulation-based method CELF [3], the subgraph-based method PMC and the sketch-based method IMM. Parameter settings are described in Table IV. Note that we did not run CELF on *Flixster* due to the prohibitively high computational costs of the simulation-based strategy in a graph of this size.

Snapshot Construction. We construct snapshots iteratively where the first snapshot is the original dataset. In each iteration, we construct the current snapshot by updating the previous one with a number of random changes and then re-assign influence probabilities for all edges of the current snapshot under the aforementioned trivalency model. More specifically, the number of random changes is equal to 30% of total number of edges



(a) NetHEPT (b) DBLP

Fig. 6: Comparison of distinct influence spreads ($k = 50$).

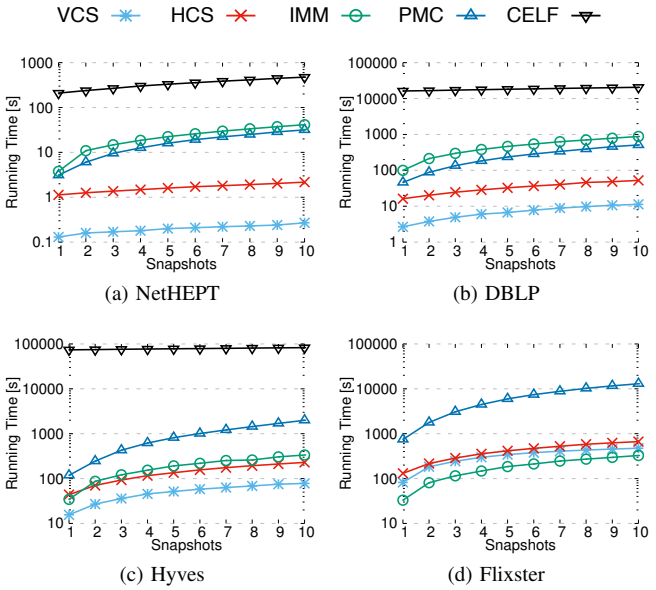
of the previous snapshot and a random change is either an edge insertion or deletion.

Distinct Influence Spread. Figure 6 compares the distinct influence spread of seed sets obtained by the five methods where the number of involved snapshots ranges from 1 to 10 and $k = 50$. Methods VCS, HCS, IMM and PMC produce similar results, and the small variations are caused by the randomness in the Monte Carlo measurements. On the other hand, CELF produces noticeably worse results but the accuracy may be improved by increasing the number of Monte Carlo simulations to estimate the influence spread.

Running Time. Figure 7 shows running time of each algorithm. CELF is the least efficient, and is around three orders of magnitude slower than HCS and VCS, and two orders of magnitude worse than PMC. The prohibitively high computational costs are caused by the requirement for a large number of Monte Carlo simulations in order to estimate the distinct influence spread of nodes across multiple graphs. PMC runs at most one and two orders of magnitude slower than HCS and VCS respectively. The relatively low efficiency is largely caused by the reduced effectiveness of its speed-up technique when extended to the DIM problem, and the costs of transforming subgraphs into acyclic graphs. IMM is around one to two orders of magnitude slower than HCS and VCS respectively on *NetHEPT* and *DBLP*. The performance overhead is mainly a result of the RR sets generated by individual snapshots, and the union sets of RR sets generated by the same node in individual snapshots. However, it outperforms PMC on *Hyves* and outperforms all methods on *Flixster*. The performance analysis of IMM for the IM problem also applies for the DIM problem.

It is worth noting that the difference between HCS and VCS w.r.t. running time becomes smaller as the graph size increases. VCS is more efficient since there exist more overlaps among subgraphs of the same snapshot than the ones among subgraphs of different snapshots, but the difference w.r.t. efficiency between them is small in large graphs, which induce more overlaps among subgraphs in different snapshots.

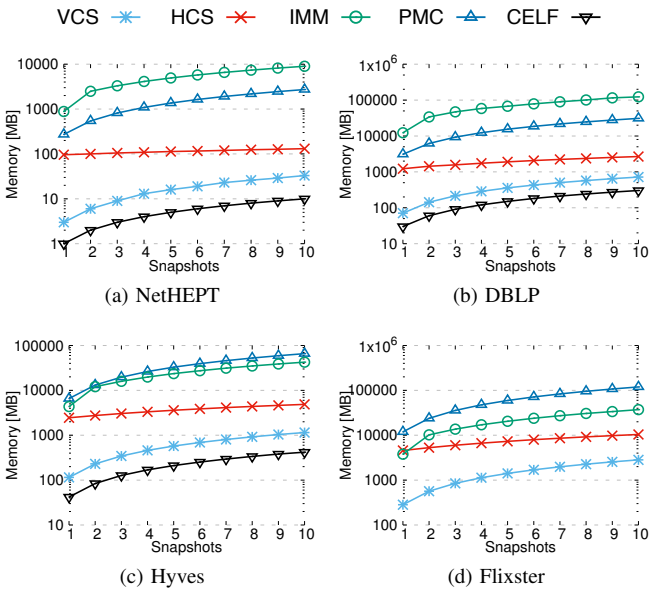
Memory Consumption. Figure 8 shows the memory consumption of each algorithm. CELF requires the least amount of memory since it only needs to store the original snapshots and estimate the influence spread using Monte Carlo simulations. Meanwhile, VCS requires an order of magnitude less memory than HCS, and two orders of magnitude less memory than



(a) NetHEPT (b) DBLP

(c) Hyves (d) Flixster

Fig. 7: Comparison of running time ($k = 50$).



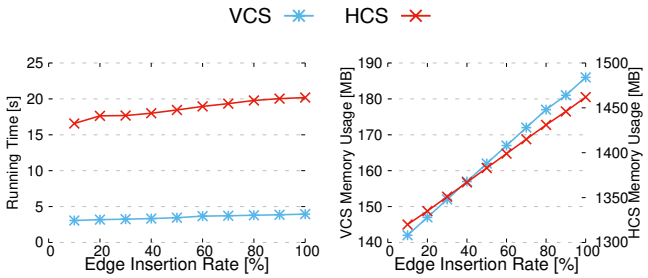
(a) NetHEPT (b) DBLP

(c) Hyves (d) Flixster

Fig. 8: Comparison of memory consumption ($k = 50$).

PMC and IMM in many cases. IMM requires more memory than PMC on *NetHEPT* and *DBLP* but outperforms PMC on larger datasets such as *Hyves* and *Flixster*. It is worth noting that the growth rate of HCS is more stable than VCS and PMC since HCS always stores a pre-defined number (i.e. 200) of horizontally-compressed graphs no matter how many snapshots are involved, and memory costs increase mainly due to non-overlapping edges and nodes from new snapshots. On the other hand, for each snapshot, VCS needs to create and store a vertically-compressed graph, and PMC needs to create a pre-defined number of subgraphs and store them separately.

Comparison between HCS and VCS. HCS is very sensitive to snapshot overlaps while VCS is more robust as snapshot differences do not influence graph compression in this technique.



(a) Efficiency Comparison (b) Scalability Comparison

Fig. 9: Performance comparison between HCS and VCS.

To better show their sensitivity to differences across snapshots, we compare the performance using two snapshots. The first snapshot used is the original *DBLP* dataset, and a second one is created by incrementally inserting 20% of the total number of edges of the original snapshot. As shown in Figure 9, as the edge insertion rate increases, the difference between running time and memory usage grows notably.

VII. Conclusions

The Influence Maximization (IM) problem is one of the most important fundamental research problems in social networks and has many well-known applications. The IM problem is NP-hard and many solutions have been proposed over the years. However, there are important real-world limitations in applications that have not been considered by previous literature. To mitigate these limitations, in this paper, we proposed and studied the Distinct Influence Maximization (DIM) problem, which aims to find a fixed seed set of k target users to maximize the expected number of distinct users influenced by the target users in an evolving social network. To solve the DIM problem, we approximate the distinct influence spread by estimating the average distinct reachability on subgraphs generated from snapshots of the social network. Due to the high memory costs of the generated subgraphs, we proposed two strategies HCS and VCS which compress the subgraphs in novel ways, and then find target users directly using the compressed graphs. In addition, we show that VCS can be used to efficiently solve the classic IM problem. Extensive experiments were performed on real-world datasets to verify the efficiency, accuracy and scalability of our solutions for the DIM and IM problem.

Acknowledgement

This work was partially supported by ARC DP170102726, DP180102050, DP170102231, and NSFC 61728204, 91646204. Zhifeng Bao is a recipient of Google Faculty Award.

References

[1] P. Domingos and M. Richardson, "Mining the network value of customers," in *SIGKDD*, 2001, pp. 57–66.
[2] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *SIGKDD*, 2003, pp. 137–146.
[3] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *SIGKDD*, 2007, pp. 420–429.
[4] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++: optimizing the greedy algorithm for influence maximization in social networks," in *WWW*, 2011, pp. 47–48.
[5] S. Cheng, H. Shen, J. Huang, G. Zhang, and X. Cheng, "Staticgreedy: solving the scalability-accuracy dilemma in influence maximization," in *CIKM*, 2013, pp. 509–518.
[6] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-i. Kawarabayashi, "Fast and accurate influence maximization on large networks with pruned monte-carlo simulations," in *AAAI*, 2014, pp. 138–144.
[7] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *SODA*, 2014, pp. 946–957.
[8] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *SIGMOD*, 2014, pp. 75–86.
[9] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *SIGMOD*, 2015, pp. 1539–1554.
[10] A. Goyal, W. Lu, and L. V. Lakshmanan, "Simpath: An efficient algorithm for influence maximization under the linear threshold model," in *ICDM*, 2011, pp. 211–220.
[11] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *ICDM*, 2012, pp. 918–923.
[12] S. Galhotra, A. Arora, and S. Roy, "Holistic influence maximization: Combining scalability and efficiency with opinion-aware models," in *SIGMOD*, 2016, pp. 1077–1088.
[13] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *SIGKDD*, 2010, pp. 1029–1038.
[14] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *SIGKDD*, 2009, pp. 199–208.
[15] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *ICDM*, 2010, pp. 88–97.
[16] S. Cheng, H. Shen, J. Huang, W. Chen, and X. Cheng, "Imrank: influence maximization via finding self-consistent ranking," in *SIGIR*, 2014, pp. 475–484.
[17] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Link mining: models, algorithms, and applications*, 2010, pp. 337–357.
[18] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *TKDD*, vol. 1, no. 1, p. 2, 2007.
[19] X. Chen, G. Song, X. He, and K. Xie, "On influential nodes tracking in dynamic social networks," in *SDM*, 2015, pp. 613–621.
[20] Y. Wang, Q. Fan, Y. Li, and K.-L. Tan, "Real-time influence maximization on dynamic social streams," *PVLDB*, vol. 10, no. 7, pp. 805–816, 2017.
[21] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-i. Kawarabayashi, "Dynamic influence analysis in evolving networks," *PVLDB*, vol. 9, no. 12, pp. 1077–1088, 2016.
[22] P. Kotler and S. J. Levy, "Broadening the concept of marketing," *The Journal of Marketing*, pp. 10–15, 1969.
[23] https://www.pwc.nl/nl/assets/documents/pwc-leading-with-customer-focused-content.pdf.
[24] http://www.adlucent.com/blog/2016/71-of-consumers-prefer-personalized-ads.
[25] https://www.clickz.com/personalization-helps-amazon-prevail.
[26] "Effective frequency: Reaching full campaign potential," White Paper, Facebook, July 2016.
[27] A. Arora, S. Galhotra, and S. Ranu, "Debunking the myths of influence maximization: An in-depth benchmarking study," in *SIGMOD*, 2017, pp. 651–666.
[28] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.
[29] J. Goldenberg, B. Libai, and E. Muller, "Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata," *Academy of Marketing Science Review*, vol. 9, no. 3, pp. 1–18, 2001.
[30] M. Lahiri and T. Y. Berger-Wolf, "Periodic subgraph mining in dynamic networks," *Knowledge and Information Systems*, vol. 24, no. 3, pp. 467–497, 2010.
[31] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.
[32] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck, "Sketch-based influence maximization and computation: Scaling up with guarantees," in *CIKM*, 2014, pp. 629–638.