Influence maximization based on reachability sketches in dynamic graphs

Dongeun Kim, Dongmin Hyeon, Jinoh Oh, Wook-Shin Han, Hwanjo Yu*

Pohang University of Science and Technology (POSTECH), Pohang, South Korea

ARTICLE INFO

Article history:
Received 22 June 2016
Revised 28 November 2016
Accepted 13 February 2017
Available online 14 February 2017

Keywords:
Social network
Influence maximization
Dynamic graph

ABSTRACT

Influence maximization is the problem of selecting the most influential nodes in a given graph. The problem is applicable to viral marketing and is actively researched as social networks become the media of information propagation. To solve the challenges of influence maximization, previous works approximate the influence evaluations to reduce the running time and to simultaneously guarantee the quality of those evaluations. We propose a new influence maximization algorithm that overcomes the limitations of the state of the art algorithms. We also devise our algorithm to process update operations of dynamic graphs. Our algorithm outperforms the state of the art algorithms TIM* and SKIM in running time, and its influence spread is also comparable to the others. Our experiments show that processing update operations is faster than executing baselines each time. Additional experiments with synthetic graphs show that the process preserves the quality of influence spread.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

In recent years, social networks have grown rapidly. People use the social networks as a place of information or opinion sharing. Because of valuable information in social networks, there have been many works for mining social networks such as [15,16].

Influence propagation in social networks is also an important factor for mining social networks from a viral marketing perspective. Users of a social network such as Facebook and Twitter share information, and some of their neighbors are influenced. The influence is propagated again to the neighbors of the influenced users, and this phenomenon spreads the information through the network. Since resources, such as budgets, are limited in viral marketing strategies, selecting influential individuals as seeds is important.

Influence maximization is a problem of selecting the most influential nodes from a given social network in consideration of influence propagation. P. Domingos and M. Richardson first pose this problem in terms of influence diffusion and propose a probabilistic model to estimate network values of customers [6,17]. Kempe et al. formalize influence maximization as a discrete optimization, which maximizes the expected number of nodes that are influenced by a set of selected seed nodes from a given graph [8]. They also propose two influence diffusion models: Independent Cascades (IC) model and Linear Threshold (LT) model, which define how influence is propagated in a given graph.

* Corresponding author.
E-mail address: hwanjoyu@postech.ac.kr (H. Yu).

http://dx.doi.org/10.1016/j.ins.2017.02.023
0020-0255/© 2017 Elsevier Inc. All rights reserved.
There are two major challenges of influence maximization for large graphs. One is that finding the optimal set of seed nodes is NP-hard. Kempe et al. propose a Greedy framework to solve the NP-hard problem and prove that it guarantees \((1 − 1/\text{e})\)-approximation of the optimal solution [8]. Leskovec et al. propose CELF greedy algorithm which reduces the number of influence evaluation with lazy update of the marginal gain for each node [12]. The other is that estimating influence spread of a node, the expected number of nodes influenced by a node, is a \#P-hard problem [3]. Based on the above greedy algorithms, the later studies focus on approximating influence spread using probabilistic methods [3,7,9] or using random sampling [2,5,18].

Among the existing algorithms, two state of the art algorithms TIM+ [18] and SKIM [5], both of which are based on random sampling, have their own limitations. TIM+ optimizes the number of samples with respect to the value of an input parameter. It has achieved the best performance as compared to other state-of-the-art approaches in running time and influence maximization. However, the TIM+ is subject to a small number of seed nodes, increasing the processing time largely. SKIM selects seed nodes incrementally using a concept of Combined Reachability Sketch, but it uses huge amounts of memory and its running time in the paper is evaluated except the preprocessing time for generating the input of the algorithm.

Another limitation of the previous works is that all of them are applicable only to static graphs. Real-world social networks, however, are expanding rapidly and are changeable in their topologies. A modification of an edge results in unreliability of the influence evaluations of the nodes near the edge because the edge may affect the evaluation of the nodes. It is challenging that influence maximization algorithm for dynamic graphs maintains influence evaluations of nodes while handling update operations.

In this paper, we propose a new influence maximization algorithm and devise the algorithm to apply on dynamic graphs. Our algorithm selects seed nodes incrementally using the similar concept to the crux of SKIM, and drops memory usage. We also invent heuristics to adjust the intermediate structures of our algorithm for processing update operations of dynamic graphs. Our experiments compare our new algorithm with baselines, and present that the process for update operations is faster than re-running baselines while maintaining the quality of influence evaluations.

2. Preliminary

Prior to our algorithm, we introduce several issues of influence maximization as preliminaries. The problem definition and challenges in static graphs are presented in Section 2.1. Independent Cascade (IC) model is described in Section 2.2 and propagation probability schemes for the model in Section 2.3, respectively. Section 2.4 revisits the greedy algorithms for influence maximization. In Section 2.5, we introduce Reverse Influence Sampling (RIS), which is an emerging algorithm for influence maximization.

2.1. Influence maximization for static graphs

The problem of selecting the most influential nodes is formalized by Kempe et al. [8]. We revisit the definition as follows.

**Definition 1.** For a given directed graph \(G = (V, E)\) and the size of a seed set \(s\), influence maximization is to find a set of seed nodes \(S\) that satisfies

\[
S = \arg \max_{S \subseteq V, |S| = s} \sigma(G, S')
\]

where \(\sigma(G, S')\) is the expected number of nodes influenced by node set \(S'\) in a graph \(G\).

The function \(\sigma(\cdot)\) is defined by an influence diffusion model, which presents how influence is propagated in a given graph. Independent Cascades (IC) model is one of simple influence diffusion model proposed by Kempe et al. [8], and IC model has been used in most of previous work [2,3,5,7-9,12,18]. Thus, we adopt IC model in our experiments to compare with baselines.

There are two challenges to solve influence maximization. The first one is that selecting the optimal seed set \(S\) that maximizes \(\sigma(G, S')\) is NP-hard and it was proven by Kempe et al. [8]. The second one is that evaluating \(\sigma(G, v)\) for a given \(v\) is \#P-hard and it was proven by Chen et al. [3]. The way widely used to evaluate influence spread of a node is the average of sufficient times (over 10,000 times) of Monte-Carlo simulations. To overcome those challenges, many previous works propose various algorithms to approximate \(\sigma(G, v)\) [3,7,9].

2.2. Independent cascade (IC) model

Before applying influence maximization to a graph, the diffusion model of a graph should be decided. Independent Cascades (IC) is a simple and famous influence diffusion model. There are some assumptions in IC model. Firstly, influence propagation is based on one to one contact between members of the graph, and Secondly, each connection between members is independent. Thirdly, the information is propagated through discrete steps.

Let \(step_t\) denote the \(t\)th step of IC model, and \(V_t\) denote the nodes that become active at \(step_t\). At first, only seed nodes are initialized to be active at \(step_0\) and the others are inactive, \(V_0 = S\). At each step \(step_t\), active nodes \(V_t\) has chances
to activate its inactive neighbors with an independent random variable through each edge. The neighbors activated by $V_t$ become $V_{t+1}$, and active nodes cannot become inactive again. Thus, each edge in the graph has at most one chance to activate its neighbors. Such cascade processes are terminated at the step where any nodes are not active, namely $V_t = \emptyset$.

2.3. Propagation probability

To assign the propagation probability to each edge, we adopt Weighted Cascade (WC) model. In WC model, the probability of each edge $p(v_{out}, v_{in})$ is assigned to $\frac{1}{\text{Degree of } v_{in}}$ where $v_{out}$ is out-node of the edge and $v_{in}$ is in-node of the edge. There are two other schemes used frequently for assigning a propagation probability to each edge: assigning a uniform probability (e.g. 0.01) to each edge, and assigning one of predefined probabilities randomly (e.g. among 0.1, 0.01, or 0.001). Among those schemes, we choose WC model for experiments because WC model outperforms two other schemes in influence spread [5,7,9].

2.4. Greedy algorithm

To overcome the NP-hardness of influence maximization, D. Kempe et al. proposes a Greedy algorithm [8]. They prove that applying the greedy framework for influence maximization guarantees $(1 - 1/e)$-approximation of the optimum if function $\sigma(\cdot)$ is non-negative, monotone and submodular. The greedy framework adds a seed node $u$ that has the maximum marginal gain of the remaining nodes,

$$u = \arg \max_{v \in V} (\sigma(G, S \cup \{v\}) - \sigma(G, S)).$$

Although Greedy algorithm offers reliable quality of influence spread, it includes huge amounts of inefficient computations. Whenever a new seed node $u$ is selected, the evaluations of influence spread for the residual nodes, $\sigma(G, v \in V \setminus u)$, require recomputations. J. Leskovec et al. observe that a new seed node affects only a small number of nodes on their influence spread [12]. Regarding their observation, they also propose CELF greedy algorithm which reduces the number of evaluations of influence spread using lazy update. To reduce the running time, the later studies focus on approximating the evaluation method, $\sigma(G, v)$, and most of them construct a probabilistic model that estimates influence spread $\sigma(G, v)$ for a given node $v$ [3,7,9].

2.5. Reverse influence sampling (RIS)

Recently, Borgs et al. propose Reverse Influence Sampling (RIS), which approximates influence spread of each node in a different way from the greedy framework [2]. RIS samples a number of random nodes in the graph and utilizes the graph in reverse directions of edges to search which nodes can influence the sampled nodes.

**Definition 2.** For a given graph $G(V, E)$ and a random sample node $v \in V$, we perform a propagation simulation from $v$ in the reverse graph $G^r$. Random Reverse Reachability Set (Random RR set) of node $v$ is a set of nodes that are reachable from $v$ in the simulation, which implies that each node $u$ in Random RR set of $v$ can reach/influence $v$ in the original graph $G$.

With a sufficient number of Random RR sets, the number of Random RR sets that contain a node $u$ is considered as the influence of $u$. In RIS, the number of Random RR set samples $R$ is determined by a formula with respect to $|V|$ and $|E|$. After generating $R$ of Random RR sets, seed nodes are selected as a maximum coverage problem.

TIM$^+$ [18] and SKIM [5] are based on RIS to evaluate influence spread of each node. The fundamental algorithm of Two-phase Influence Maximization (TIM) is same to RIS. However, TIM$^+$ optimizes the number of Random RR sets with respect to a parameter $\epsilon$ and a small number of sampled Random RR sets. On the other hand, Sketch-based Influence Maximization (SKIM) selects seed nodes incrementally with a fixed number of propagation instances which are defined in the following section. SKIM estimates Combined Reachability Sketches which approximate the size of Combined Reachability Set for each node. Combined Reachability Sketch and Combined Reachability Set are also defined in the following section.

2.6. Combined reachability sketch

The input of SKIM is a fixed set of $l \geq 1$ propagation instances, $(G^{(i)})$ where $G^{(i)} = (V, E^{(i)})$. Such propagation instances are obtained by applying a random variable to each edge and getting the subset of edges. Combined Reachability Set is defined in **Definition 3**.

**Definition 3.** The Combined Reachability Set of a node $u$, $R_u$, is a set of node-instance pairs that is reachable,

$$R_u = \{(v, i)|u \sim_{G^{(i)}} v\}$$

where $u \sim_{G^{(i)}} v$ means node $v$ is reachable from $u$ along the edges of $G^{(i)}$.

SKIM approximates $\sigma(G, S)$ based on Combined Reachability Sketches which estimate the sizes of Combined Reachability Sets. Each node-instance pair $(v, i)$ is associated with a random rank $r_{v}^{(i)} \sim U[0, 1]$, and bottom-k min-hash sketches are used for Combined Reachability Sketches [4].
Definition 4. Given the size of sketches $k$, the Combined Reachability Sketch of a node $u$, $X_u$, is a set of $k$ smallest rank values,

$$X_u = \text{Bottom-}k\{r_v^{(i)}|(v, i) \in R_u\}$$

where $\text{Bottom-}k\{\cdot\}$ is $k$ smallest values of a set $\{\}$.

Let the threshold rank $\tau_u$ of a node $u$ is the $k$-th rank of $X_u$. Then, we can estimate the size of Combined Reachability Set of a node $u$ $|R_u|$ as $(k-1)/\tau_u$ if $|X_u| = k$ and as $|X_u|$ if $|X_u| < k$. Cohen et al. [5] prove the correctness of Combined Reachability Sketch empirically.

3. New algorithm for influence maximization

In this section, we propose a new algorithm for influence maximization. We discuss the limitations of two state of the art algorithms, TIM$^+$ and SKIM, in Section 3.1. To overcome the limitations of the algorithms, we define reachability sketch using reverse sampling in Section 3.2. Finally, our new algorithm is described in Section 3.3.

3.1. Limitations of existing algorithms

Previous works for influence maximization in static graphs still have several weak points. TIM$^+$ shows the finest performance so far, but the process for selecting seed nodes is not incremental. In other words, after the optimized number of Random RR sets are generated, TIM$^+$ starts to select seed nodes but the order of selecting seed nodes does not guarantee the order of influence spread. Another weak point of TIM$^+$ is that it runs slower with the smaller size of seeds $s$. The $\epsilon$ value of TIM$^+$, which affects the number of sample nodes, is even one of input parameters. On the other hand, SKIM shows slightly worse performance than TIM$^+$, but the difference is insignificant and SKIM finds seed nodes incrementally whereas TIM$^+$ does not. Nevertheless, SKIM has two crucial limitations: it requires a large amount of memory due to $l$ propagation instances, and it assumes that a set of propagation instances, which is the input of SKIM, is already provided. If we consider the time for generating propagation instances, the running time of SKIM becomes much slower.

3.2. Reachability sketch using reverse sampling

Cohen et al. [5] suggest the way to build sketches without the propagation instances. Assuming that the number of propagation instances $l \rightarrow \infty$ allows us to fill the sketches generating random cascades from random sample nodes. In their paper, however, the idea of building sketches without propagation instances is described abstractly, and they propose the algorithm using propagation instances though.

We suggest Reachability Sketch as a more general concept of Combined Reachability Sketch for overcoming the weak point of SKIM in the memory usage. SKIM requires $l$ propagation instances as its inputs which result in the high spatial complexity. The size of a propagation instance $G^{(i)}$, which is a subgraph of the original graph $G$, is proportional to the size of $G$. Nevertheless, we notice the concept of Combined Reachability Sketch for the approximation of the influence of nodes. For these reason, we define Reachability Sketch as the approximation of the influence of nodes for our algorithm.

We devise new influence maximization algorithm, Reachability Sketch using Reverse Sampling (RSRS). In our method, a random node is selected only over the original given graph. In order to substitute the propagation instances of SKIM, we allow redundant selecting of a node in the given graph and give a unique value to redundant nodes. As a result, memory usage can be dropped to the original graph size.

Our Reachability Sketches are built using Random RR sets and described in Definition 6. First, we assign a unique number to each random sample node $v$ for identification. Next, we associate each node-id pair $(v, c)$ with a random rank $U[0, 1]$. Then, we build Random RR sets from node-id pairs $(v, c)$, and Reachability Sketches are filled with the ranks of the pairs $r_{v^c}$. We revisit RR set in Definition 5.

Definition 5. Let $S(v, c)$ is a propagation simulation by using Breath-First Search (BFS) from a sample node-id pair $(v, c)$ in a reverse graph $G^T$. Reverse Reachability Set (RR set) of $(v, c)$, $RR_{v^c}$, is a set of nodes that is reachable from node $v$ in $S(v, c)$,

$$RR_{v^c} = \{u|u \sim_{S(v, c)} v\}$$

where $u \sim_{S(v, c)} v$ means node $v$ is reachable from $u$ in the propagation simulation $S(v, c)$.

Definition 6. Given the size of sketches $k$, Reachability Sketch using Reverse Sampling (RSRS) of a node $u$, $RS_u$, is a bottom-$k$ min-hash sketch of the rank values $r_{v^c}$ where $v$ is reachable from $u$ in $S(v, c)$,

$$RS_u = \text{Bottom-}k\{r_{v^c}|u \sim_{S(v, c)} v\}$$

(6)
Algorithm 1 RSRS($G(V, E)$, s, k, r).

1: for all nodes $v$ do
2: \hspace{0.5cm} covered[$v$] ← false
3: \hspace{0.5cm} size[$v$] ← 0
4: \hspace{0.5cm} count[$v$] ← 0
5: \hspace{0.5cm} $rr$ ← hash map of node-count pairs to nodes
6: \hspace{0.5cm} seedset ← $\emptyset$
7: \hspace{0.5cm} sequence ← $\emptyset$
8: while $|seedset| < s$ do
9: \hspace{1cm} building ← true
10: \hspace{1cm} while building do
11: \hspace{1.5cm} $v$ ← random($V$)
12: \hspace{1.5cm} count[$v$] ← count[$v$] + 1
13: \hspace{1.5cm} $c$ ← count[$v$]
14: \hspace{1.5cm} sequence.add($v$, $c$)
15: \hspace{1.5cm} if covered[$v$] = false then
16: \hspace{1.75cm} Simulate using BFS from $v$ in $G$
17: \hspace{1.75cm} for visited node $u$ during BFS do
18: \hspace{2.25cm} size[$u$] ← size[$u$] + 1
19: \hspace{2.25cm} $rr$[$v$, $c$] ← $rr$[$v$, $c$] $\cup$ [$u$]
20: \hspace{2.25cm} if size[$u$] = $k$ then
21: \hspace{2.75cm} $x$ ← $u$
22: \hspace{2.25cm} building ← false
23: end while
24: if size[$u$] < $k$ for all nodes $u$ then
25: \hspace{0.5cm} $x$ ← argmax$_{u \in V}$size[$u$]
26: \hspace{0.5cm} for $i = 1$ to $r$ do
27: \hspace{1cm} Simulate using BFS from $x$ in $G$
28: \hspace{1cm} for visited node $v$ during BFS do
29: \hspace{1.5cm} covered[$v$] ← true
30: \hspace{1.5cm} for all nodes $u$ in $rr$[$v$, $c$] where $1 \leq c \geq count[v]$ do
31: \hspace{2cm} size[$u$] ← size[$u$] − 1
32: \hspace{2cm} erase $rr$[$v$, $c$] where $1 \leq c \leq count[v]$
33: \hspace{1.5cm} end for
34: \hspace{1cm} seedset.add($x$)
35: end while

3.3. Influence maximization using RSRS

Algorithm 1 shows the pseudo code of our algorithm using RSRS. We assume that the ranks of sampled nodes are assigned in ascending order similarly to SKIM. Thus, the first seed node is a node $u$ whose $RS_u$ is filled earliest. For a new seed node, we simulate $r$ times in the same manner of D. Kempe et al. [8], and the visited nodes are set to covered such that covered nodes are no longer sampled. The RSRS is implemented as an inverted index with the size of RSRS for each node, size[-], and a hash map of Random RR sets, $rr[-, -]$. The id of each node-id pair is simply assigned to the counts of a node.

In Algorithm 1, there are three additional variables that are not defined before: $\mathcal{E}^{c}_{rs}$, sequence, and scanned rank. These variables are used for handling update operations of dynamic graphs. The active edges are stored in $\mathcal{E}^{c}_{rs}$ during BFS in $G^r$ for each pair $(v, c)$, but this variable is not presented in Algorithm 1 explicitly. sequence keeps the permutation of sampled node-id pairs, and scanned rank keeps the current index of the sequence as the rank $r^{c}$. The details are described in Section 4.3.

4. Influence maximization in dynamic graphs

The existing algorithms solve Influence Maximization only for static graphs. However, the typical application of influence maximization is viral marketing on social networks which are dynamically changing in their topologies. Most of the literatures use a random variable or a probabilistic model for estimating the influence spread. Thus, any modifications of the target graph make the evaluations of the influence spread unreliable. We propose a framework to handle the dynamic graph in influence maximization. The heart of the framework is checking whether the update operations can affect the built sketches or not. Because the approximation scheme keeps the sketches of nodes as mentioned in Section 3.2.
4.1. Entire process for influence maximization in dynamic graphs

We propose a framework for processing dynamic graphs using our new influence maximization algorithm. Fig. 1 shows three phases of the entire process. At the initialization phase, sketches are built until $\max(|X_u|) = k$ where $X_u$ is a sketch of node $u$ and $k$ is the size of sketch, which means that the first seed nodes are found.

The framework adjusts the partially built sketches for the update operations, or continues building sketches for an influence maximization query as presented in Section 3.3. Section 4.2 defines dynamic graphs and additional notations, and Section 4.3 describes the process for update operations.

4.2. Problem definition for dynamic graphs

In this section, we first present additional notations for dynamic graphs and an assumption to solve influence maximization in dynamic graphs. Real world graphs such as social networks are changing dynamically. Their topologies are modified by update operations such as insertions/deletions of nodes/edges.

**Definition 7.** A dynamic graph $\mathcal{G}$ is a sequence of updates

$$\mathcal{G} = \{U_1, ..., U_z\}$$

where $U_i \in \mathcal{G}$ is one update operation that inserts a node or an edge, or deletes a node or an edge and $z$ is the number of updates.

Let us consider the effect of isolated nodes on influence maximization. Since information is propagated through edges in social networks, an isolated node cannot influence other nodes and be influenced by other nodes. For this reason, we revise the definition of updates as follows. Process for an insertion of a new node is delayed until an edge that is incident to the new node is inserted. A deletion of a node that has edges is converted to deletions of those edges.

**Definition 8.** An update operation $U_t$ consist of an edge and an operator,

$$U_t = \langle (v_{\text{out}}, v_{\text{in}}), \text{op} \rangle$$

where $v_{\text{out}}$ is out-node of the edge, $v_{\text{in}}$ is in-node of the edge, and $\text{op}$ is the operator which is either insert or delete.

We define two additional notations of methods for dealing with update operations. The method simulate($v_{\text{out}}$, $v_{\text{in}}$) simulates the edge ($v_{\text{out}}$, $v_{\text{in}}$) with respect to the assigned probability and returns a boolean value whether the edge becomes active or not. There is the other method scanGT($u$, $\mathcal{R}_u^c$, $\mathcal{G}$) that is frequently used for processing updates. scanGT($u$, $\mathcal{R}_u^c$, $\mathcal{G}$) performs a simulation using BFS from node $u$ in $\mathcal{G}$ and fills $\mathcal{R}_u^c$, which is equivalent to the line 16–19 in Algorithm 1 if $u = v$.

4.3. Process for update operations

As defined in Section 4.2, there are three operations for influence maximization in dynamic graphs: an insertion of a new node, an insertion of an edge, and a deletion of an edge. Thus, the outline of the process for an update operation $U_t$ is as follows. The process for a new node NewNode($U_t$, $(v_{\text{out}}, v_{\text{in}})$) and the process for an insertion of an edge InsertEdge($U_t$, $(v_{\text{out}}, v_{\text{in}})$) are performed if $U_t$.op is insert. On the other hand, only the process for a deletion of an edge DeleteEdge($U_t$, $(v_{\text{out}}, v_{\text{in}})$) is performed if $U_t$.op is delete. The rest of this section describes the process for each operation and integration of those processes.

![Fig. 1. Three process of influence maximization in dynamic graphs.](image-url)
4.3.1. Insertion of a new node

We should consider a new node in two possible effects on the built sketches. One possible effect is that the new node should have been sampled while building sketches. First, we should compute how many times the new node should have been sampled until current scanned rank. A random rank is assigned to each sample of the new node. Next, scanGT(v, RR_i^G, G) should be performed for each sample (v, c) which has a lower rank than scanned rank. Note that the maximum number of samples is \( n \times k \) where \( n \) is the number of nodes in given graph and \( k \) is the size of sketches. Fig. 2 represents the process. The other possible effect of a new node is that the sketch of the new node could have been built by nodes sampled before the current process. Applying the effect on the built sketches, however, could be handled as an insertion of an edge because the insertion of an edge that is incident to a new node occurs the operation of the new node. Algorithm 2 presents the pseudo code that applies the first effect.

![Diagram](image)

**Fig. 2.** The example of newnode \( v_i \).

**Algorithm 2** NewNode(\( v_{out}, v_{in} \)).

1. NewNodes \( \leftarrow \emptyset \)
2. if \( v_{out} \) is new then
3. NewNodes \( \leftarrow \) NewNodes \( \cup v_{out} \)
4. if \( v_{in} \) is new then
5. NewNodes \( \leftarrow \) NewNodes \( \cup v_{in} \)
6. for all \( v \in \) NewNodes do
7. for \( c = i \) to \( k \) do
8. \( r_i^c \leftarrow \text{Rand}(1, n \times k) \)
9. if \( r_i^c < \text{scanned rank} \) then
10. scanGT(v, RR_i^G, G)

4.3.2. Insertion of an edge

For an insertion of an edge, we first distinguish which sketches are affected by the insertion. There are three cases of the reverse reachability sketch RR_i^G of a pair \((u, c)\) for the insertion of an edge \((v_{out}, v_{in})\).

1. \( v_{in} \notin RR_i^G \): It means that scanGT(u, RR_i^G, G) did not activate \( v_{in} \), and the insertion of \((v_{out}, v_{in})\) cannot affect \( RR_i^G \).
2. \( v_{out} \in RR_i^G \) and \( v_{in} \in RR_i^G \): In this case, we just perform simulate(\( v_{out}, v_{in} \)) to decide whether \( (v_{out}, v_{in}) \) is active in \( RR_i^G \). The insertion, however, cannot affect the sketch because \( v_{out} \) can be active only once due to the definition of IC model.
3. \( v_{out} \notin RR_i^G \) and \( v_{in} \in RR_i^G \): This case may expand the built sketches. First, perform simulate(\( v_{out}, v_{in} \)). If \((v_{out}, v_{in})\) becomes active, \( v_{in} \) also becomes active and BFS is performed from \( v_{in} \) in \( G^T \) to add the visited nodes to \( RR_i^G \), scanGT(\( v_{in}, RR_i^G, G \)). The above process is presented in Algorithm 3.

**Algorithm 3** InsertEdge(\( v_{out}, v_{in} \)).

1. for all \( RR_i^G \) do
2. if \( v_{in} \in RR_i^G \) then
3. active \( \leftarrow \) simulate(\( v_{out}, v_{in} \))
4. if active then
5. \( E_i^G \leftarrow E_i^G \cup (v_{out}, v_{in}) \)
6. if \( v_{out} \notin RR_i^G \) then
7. scanGT(\( v_{in}, RR_i^G, G \))
4.3.3. Deletion of an edge

As considered in the similar way to insertions of edges, only the case that \( v_{\text{out}} \in R_R^u \) and \( v_{\text{in}} \in R_R^u \) affects the built sketches in the deletion. Unreachable nodes by deleting an edge should also be deleted in \( R_R^u \) for each \( u \). We introduce dependent and independent variables which are dependent and independent nodes with a deletion of the edge in \( R_R^u \), respectively. When \( v_{\text{out}} \) is added to \( R_R^u \) through edge \((v_{\text{out}}, v_{\text{in}})\) and cannot be added through other incident edges to \( v_{\text{out}} \). \( v_{\text{out}} \) is dependent on \((v_{\text{out}}, v_{\text{in}})\) in \( R_R^u \). We facilitate the active edges \( \mathcal{E}_u^c \) of sketches to classify the dependent nodes on the edge \((v_{\text{out}}, v_{\text{in}})\) in \( R_R^u \).

Algorithm 4 presents the process of a deletion of an edge. First, remove edge \((v_{\text{out}}, v_{\text{in}})\) from active edges \( \mathcal{E}_u^c \) to find nodes independent with current deletion operation. And then, perform BFS from \( u \) in \( G(V, \mathcal{E}_u^c)^T \) only using \( \mathcal{E}_u^c \) without their associated probabilities. Next, the nodes visited are not dependent on \((v_{\text{out}}, v_{\text{in}})\) among \( R_R^u \). Finally, remove dependent nodes and their incident edges from \( R_R^u \) and \( \mathcal{E}_u^c \), respectively.

4.3.4. Integration of update process

The three update processes can be combined into one method. Algorithms 2–4 have the loop statement for all \( R_R^u \) in common. Algorithm 5 shows the integration code.

Algorithm 5 AdjustSketches \((v_{\text{out}}, v_{\text{in}}, \text{op})\).

\[
\begin{array}{ll}
1: & \text{for all } R_R^u \text{ do} \\
2: & \quad \text{if } v_{\text{out}} \in R_R^u \text{ and } v_{\text{in}} \in R_R^u \text{ then} \\
3: & \quad \quad \mathcal{E}_u^c \leftarrow \mathcal{E}_u^c - (v_{\text{out}}, v_{\text{in}}) \\
4: & \quad \quad \text{independent} \leftarrow \text{BFS from } u \text{ in } G(V, \mathcal{E}_u^c)^T \\
5: & \quad \quad \text{dependent} \leftarrow (R_R^u - \text{independent}) \\
6: & \quad \quad \text{RR}_u \text{.remove}(\text{dependent nodes & incident edges}) \\
7: & \quad \text{if op = insert then} \\
8: & \quad \quad \text{active} \leftarrow \text{simulate}(v_{\text{out}}, v_{\text{in}}) \\
9: & \quad \quad \text{if active then} \\
10: & \quad \quad \quad \mathcal{E}_u^c \leftarrow \mathcal{E}_u^c \cup (v_{\text{out}}, v_{\text{in}}) \\
11: & \quad \quad \quad \text{RR}_u \text{.remove}(\text{dependent nodes & incident edges}) \\
12: & \quad \text{else} \\
13: & \quad \quad \quad \text{if } v_{\text{out}} \in R_R^u \text{ and } v_{\text{in}} \in R_R^u \text{ then} \\
14: & \quad \quad \quad \quad \text{independent} \leftarrow \text{BFS from } u \text{ in } G(V, \mathcal{E}_u^c)^T \\
15: & \quad \quad \quad \text{dependent} \leftarrow (R_R^u - \text{independent}) \\
16: & \quad \quad \text{RR}_u \text{.remove}(\text{dependent nodes & incident edges}) \\
\end{array}
\]

The Algorithm 2, 3 and 4 are represented in line 2–4, 5–11 and 12–16 of Algorithm 5 respectively. Line 1–9 of the Algorithm 2 are still used to find NewNodes and to set the flag NewFlag whether there are any new nodes prior to Algorithm 5.

5. Experiments

5.1. Experiment setup

We perform three kinds of experiments for the performance of our influence maximization algorithm and the performance of the process for update operations comparing to baselines. Firstly, we compare RSRS with SKIM and TIM+ in terms of running time, influence spread, and memory usage. Secondly, we generate dynamic datasets and perform our framework for dynamic graphs. Finally, we also generate synthetic datasets to present the guarantee of the update process.

As in the experiment parts of TIM+ and SKIM, we set the parameter \( \epsilon \) of TIM+ to 1, and the size of sketches \( k \) to 64 of SKIM, and the number of propagation instances \( l \) to 64 of SKIM. In the case of RSRS, the size of sketches \( k \), which manages the approximation of influence spread, is set to 64 which is equal to that of SKIM. The number of simulations \( r \) for residual
process in RSRS is set to 16, but the differences in performance between various values of r are negligible. Influence spread of output seed nodes from each algorithm is evaluated by 20,000 times of random cascades.

The original datasets are Stanford [13], DBLP [14], Patents [11], and LiveJournal [1]. All those datasets are available in SNAP [10]. Table 1 shows the details of the original datasets. There are many previous works on dynamic graphs in other fields, but we could not find any datasets that includes both insertion and deletion operations. For this reason, we generate dynamic graphs based on those original graphs. The details of generated datasets are presented in the below sections.

All the algorithms are implemented in C++. All experiments are performed on a Linux, CentOS 6.3 x86_64, machine with total 12 cores of Inter Xeon CPU X5650 2.67 GHz and 24 GB of memory. We run TIM+ with the implementation of the authors, but we implement SKIM for ourselves due to the proprietary problem of their implementation. For the experiments for dynamic graphs, several kinds of components are implemented on the two baselines.

### 5.2. RSRS

This section compares the performance of RSRS to those of the existing algorithms. The baselines are SKIM [5] and TIM+ [18], both of which are based on Reverse Influence Sampling (RIS). Figs. 3 and 4 present the results of RSRS and baselines on the original datasets shown in Table 1. RSRS outperforms SKIM and TIM+ in running time and provides the comparable performance of influence spread. The result of SKIM in LiveJournal is omitted due to its memory usage problem.

We report two cases of SKIM in running time. SKIM(G) includes the time for generating propagation instances which is the input of SKIM, and SKIM(X) does not includes that time. Even though the results of SKIM(X) is close to those of the original paper, our algorithm outperform SKIM(X) in running time.

Fig. 5 compares memory usage of three algorithms. SKIM uses huge amounts of memory compared to others because the size of propagation instances are proportional to the size of the graph. Memory usage of RSRS and TIM+ looks similar to the results of their running time because their running time and memory usage are related to the number of sample nodes. In addition, the memory usage of TIM+ decreases with a small k and keeps increasing with a bigger k. The reason is a design issues of an equation which selects the number of random RR sets in the node selection phase of TIM+.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Stanford</th>
<th>DBLP</th>
<th>Patents</th>
<th>LiveJournal</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Nodes</td>
<td>282K</td>
<td>655K</td>
<td>3775K</td>
<td>4848K</td>
</tr>
<tr>
<td># of Edges</td>
<td>2312K</td>
<td>3980K</td>
<td>16,519K</td>
<td>68,994K</td>
</tr>
</tbody>
</table>

**Fig. 3.** Running time of TIM+, SKIM, and RSRS.
Fig. 4. Influence spread of TIM\(^+\), SKIM, and RSRS.

Fig. 5. Memory usage of TIM\(^+\), SKIM, and RSRS.
Table 2
The generated dynamic datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Stanford</th>
<th>DBLP</th>
<th>Patents</th>
<th>LiveJournal</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Nodes</td>
<td>282K</td>
<td>614K</td>
<td>3775K</td>
<td>4848K</td>
</tr>
<tr>
<td># of Edges</td>
<td>2197K</td>
<td>3781K</td>
<td>16,354K</td>
<td>68,828K</td>
</tr>
<tr>
<td># of ops</td>
<td>139K</td>
<td>239K</td>
<td>206K</td>
<td>207K</td>
</tr>
</tbody>
</table>

Fig. 6. Running time and influence spread for Stanford.

5.3. Experiment on dynamic datasets

As mentioned in Section 5.1, we compare our proposed algorithm for dynamic graphs to baselines with dynamic datasets. The dynamic datasets are generated based on random selections. We select edges to insert randomly, and separate those edges from original datasets. We shuffle the edges to insert, and construct incremental dynamic datasets. On the generated incremental dynamic datasets, random edges are selected as edges to delete at random time. Update streams consist of the edges to insert and to delete which are selected as the above process. Table 2 shows the generated dynamic datasets.

We suppose that the various number of update operations are combined as the form of batches, and we set the size of batches to 50 and 500. Three algorithms repeated processing a batch of updates and a query of influence maximization. The running time of each algorithm is measured as follows.

1. TIM+: (time to update graphs) + (time to solve influence maximization)
2. SKIM: (time to update propagation instances) + (time to solve influence maximization)
3. Our algorithm (DyIM): (time to update graphs) + (time to process update operations) + (time to solve influence maximization)

Since the initial sketches in our algorithm is built once, time to build those is excluded from running time measurement. For the same reason, the time for generating propagation instances in SKIM is also excluded.

To measure influence spread of each algorithms, we average influence spread of nine points on running time. The picked points are set with an interval proportional to the number of update operations in the streams. Due to the time cost of this process, we conduct experiments in this section up to 50 seeds.

Figs. 6–9 compare our algorithm for dynamic graphs with TIM+ and SKIM. We omit the results of SKIM on Patents and LiveJournal due to the excessive memory usage for its propagation instances. Our Algorithm (DyIM) outperforms TIM+ and
Fig. 7. Running time and influence spread for DBLP.

Fig. 8. Running time and influence spread for Patents.
SKIM in running time. In influence spread, however, TIM+ provides slightly better performance than our algorithm because of the difference of approximate techniques between TIM+ and RSRS. Nevertheless, RSRS selects seed nodes incrementally in order of influence whereas TIM+ does not, and TIM+ shows slower running time with the decrease in the number of seed nodes s unless users adjust the parameter $\epsilon$ of TIM+ manually. Moreover, the process for update operations we proposed can be applied to TIM+ with some trivial tuning.

5.4. Experiment on synthetic datasets

We perform our algorithm on synthetic datasets to confirm whether the process for update operations preserves the quality of influence maximization. We observe that the output seed nodes are rarely changing during the experiments on dynamic datasets, which means that the small number of edges cannot affect the influence evaluations of significant nodes. For this reason, we generate synthetic datasets as follows: (1) pick 50 of seed nodes using RSRS (2) remove the incident edges to those seed nodes (3) add the incident edges of seed nodes to the insertion list. In other words, the original seed nodes and their incident edges are removed from the initial graphs, and they are added to the graphs dynamically. Table 3 presents the generated synthetic data.

Fig. 10 provides the results on synthetic datasets. As the update operations are processed, the influence spread becomes closer to the influence spread of the original datasets. It proves that the process for update operations adjusts the sketches correctly.

6. Conclusion

We propose a new algorithm for influence maximization, RSRS. It outperforms two state of the art algorithms, TIM+ and SKIM, and overcomes their limitations. We also suggest the process for update operation of dynamic graphs, which
preserves influence spread and shows faster running time than running an algorithm for influence maximization over again. The process is also applicable to TIM* and SKIM with some suitable tuning.

Acknowledgement

This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1401-04.

References


