Influence Analysis in Evolving Networks: A Survey

Yu Yang and Jian Pei, *Fellow, IEEE*

Abstract—Influence analysis aims at detecting influential vertices in networks and utilizing them in cost-effective business strategies. Influence analysis in large-scale networks is a key technique in many important applications ranging from viral marketing and online advertisement to recommender systems, and thus has attracted great interest from both academia and industry. Early investigations on influence analysis often assume static networks. However, it is well recognized that real networks like social networks and the web network are not static but evolve rapidly over time. Thus, to make the results of influence analysis in real networks up-to-date, we have to take network evolution into consideration. Incorporating evolution of networks into influence analysis raises many new challenges, since an evolving network often updates at a fast rate and, except for the network owner, the evolution is usually even not entirely known to people. In this survey, we provide an overview on recent research in influence analysis in evolving networks, which has not been systematically reviewed in literature. We first revisit mathematical models of evolving networks and commonly used influence models. Then, we review recent research in five major tasks of evolving network influence analysis. We also discuss some future directions to explore.

Index Terms—Influence diffusion, influence analysis, evolving networks

1 Introduction

Influence diffusion is a fundamental process taking place in networks, which refers to the behavioral change of individuals affected by others in a network. The higher the influence of an individual or a group of individuals, the more others in the network can be influenced by the individual or the group. In the past decade, influence analysis has become one of the most important research topic in network and graph mining, due to its wide applications in viral marketing [1], [2], [3], [4], recommender systems [5], [6], source detection [7], [8], outbreak monitoring [9], [10], finding leaders in communities [11], [12], optimization of network topology [13], [14], identifying key blog posts to read [15], [16], diffusion minimization/immunization [17], [18], competing diffusion analysis [19], diffusion threshold analysis [20], etc. The core of influence analysis lies at measuring influence of vertices and extracting influential vertices in large-scale networks.

Evolving network analysis has drawn substantial attention from researchers, since most real world networks, such as online social networks and web graphs/networks, are constantly evolving over time [21], [22]. Possible updates of an evolving network include insertion and deletion of vertices (objects), insertion and deletion of edges (relationships), and modifications of edge weights (strength of relationships). Some network analysis tasks in evolving networks, such as dense subgraph detection [23], [24], [25], [26], anomaly detection [27], [28], [29], and betweenness centrality computing [30], [31], [32], have been extensively investigated recently.

It is well known that influence of vertices in a network highly depends on the structure and edge weights in the network. Early studies in influence analysis all assume static networks. However, almost all networks where influence diffusion takes place, such as online social networks like Facebook [21], [22] and Twitter [33], are not static but evolve rapidly all the time. Thus, the results computed by traditional influence analysis algorithms become stale over time quickly. To maintain the freshness of the end results, we have to consider evolution of real networks. As a result, influence analysis in evolving networks has found an increasing interest in both practice and literature.

Incorporating network evolution into influence analysis raises a few computational challenges. For example, rapidly evolving networks require that influence analysis algorithms have to take into account the changes of a network and then quickly update the results obtained from early snapshots of the network. Therefore, it is desirable to provide methods that can maintain results in an incremental manner. Moreover, note that the evolution of a network can be only partially known or even hidden to us. For example, for online social networks like Twitter, except for the Twitter Company, it is very difficult for others to have the complete information of the evolution of the Twitter network. People normally can only access the evolution of a few users in Twitter by monitoring them with open APIs.

To the best of our knowledge, no existing surveys have systematically reviewed literatures in evolving network influence analysis. Sun et al. [34] and Adrien et al. [35] did

- *Y. Yang is with the School of Data Science, City University of Hong Kong, Hong Kong, China. E-mail: yuyang@cityu.edu.hk.*
- *J. Pei is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. E-mail: jpei@cs.sfu.ca.*

Manuscript received 5 May 2018; revised 12 July 2019; accepted 29 July 2019. Date of publication 14 Aug. 2019; date of current version 3 Feb. 2021. (Corresponding author: Yu Yang.)
Recommended for acceptance by L. Chen.
Digital Object Identifier no. 10.1109/TKDE.2019.2934447

surveys majorly on how to model influence diffusion in static networks. Li et al. [36] thoroughly reviewed algorithmic techniques for the well-known influence maximization problem, where only a couple of studies in evolving network influence maximization are discussed. Charu and Karthik [33] provided an overview of the vast literature on graph evolution analysis and the numerous applications that arise in different contexts. However, influence analysis in evolving networks is not covered by [33].

In this survey, we categorize recent progress along the line of influence analysis in evolving networks into five major tasks, where a number of algorithmic challenges arise. In the first three tasks, we have the complete information of network evolution, and want to identify and maintain influential vertices efficiently in an evolving network, which is one of the most important goals of influence analysis. The last two tasks are both about detecting and learning the evolution, since as illustrated above, we often do not have the complete information about network evolution.

Below, we briefly illustrate the five major tasks and discuss their relation.

1.1 With Complete Information of Network Evolution

When we have the complete information about the evolution of a network, we are interested in how to extract influential vertices from this evolving network.

1.1.1 Maintaining Individual Influence in Evolving Networks

Given network evolution, which is represented by a stream of network snapshots or a stream of updates, how can we maintain top influential individual vertices in the latest snapshot? Since real networks often evolve fast, it is crucial to design incremental algorithms that can efficiently update the maintained influential individuals against small changes of networks. The key point of this task is to maintain the influence of each individual vertex in the network in an online manner, or in other words, incrementally.

This task may be useful in applications like recommendation of influential twitters or bloggers in an evolving social network. By following influential individuals, a user can obtain hot postings that are widely spread in the network as soon as possible. Moreover, individual influence of a vertex is often used as an important feature in graph-based machine learning tasks.

1.1.2 Maintaining Influential Seed Set in Evolving Networks

When we have the complete information about the network evolution, how can we maintain a set of influential users, called a seed set, whose combined influence in the latest snapshot is as high as possible? Extracting influential seed set from a static network has been extensively studied. Under the evolving network setting, incrementally maintaining influence of vertices is a key problem. Moreover, how to utilize the maintained influence of vertices to efficiently extract an influential seed set in the latest snapshot, under the evolving network setting, is also a crucial building block. Note that extracting a seed set with high combined influence is a combinatorial optimization problem that is much more difficult than just returning top vertices with high individual influence spreads.

This task can be used to, for example, provide users real-time viral marketing plans in a highly dynamic evolving social network.

1.1.3 Extracting Influential Vertices from Temporal Networks

In the above two tasks, influence of vertices is defined as how influential they are in the latest snapshot of the evolving network. This definition is inherited from traditional influence analysis in static networks, because the latest snapshot is also a static network. Thus, we call this kind of influence the *static influence*.

Besides static influence, influence of vertices can also be defined as the aggregation of how influential they were in a past period, which is called the *temporal influence*. Comparing to static influence, temporal influence is more suitable for applications involving networks that evolve at very high rates. This is because influence diffusions also take time and, during a diffusion, the network may evolve. In such a case, finding vertices who were influential recently (with high temporal influence) may be more informative. Given a temporal network, how to build a temporal diffusion model that captures the intuition of temporal influence and how to extract influential vertices accordingly are important problems along this line.

1.2 Without or With Partial Information of Network Evolution

When we do not have the complete knowledge about the network evolution, we need to detect and learn the evolution as much as possible in order to conduct effective influence analysis.

1.2.1 Probing Network Evolution for Influence Analysis

Sometimes although we do not have the complete information about the evolution of a network, we can monitor some selected vertices and detect their evolution. However, it is usually infeasible to monitor all vertices. For example, Twitter has APIs for crawling connections of users, but these APIs all have access limits. Thus, usually we can only choose a relatively small number of users to monitor their evolution. Since our goal is to conduct effective influence analysis, we need to focus on the vertices whose evolution may affect influence diffusions the most. How to evaluate the importance of a vertex's evolution to influence diffusions in the network, and decide which vertices to monitor accordingly are the key problems in this task.

This task plays an important role in providing the input data to the first three tasks when the network is not entirely known to the one who wants to conduct influence analysis.

1.2.2 Learning Up-to-Date Influence Weights

In influence analysis, influence weights, which are weights assigned to edges that reflect strength of peer influence of adjacent users, are very important in deciding users' influence. By probing a vertex, we get its current neighbors

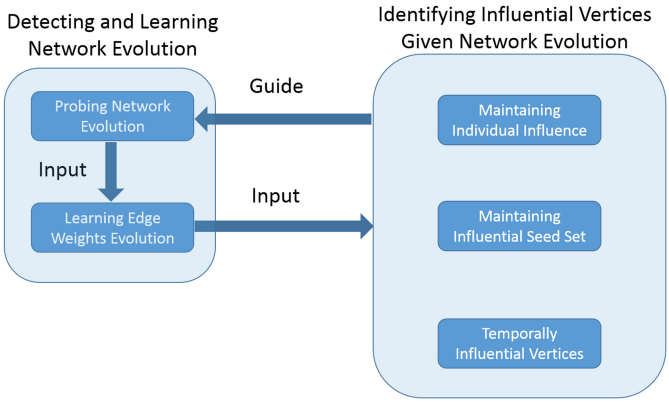


Fig. 1. Connections of five major tasks.

TABLE 1
Representative Studies Introduced in this Survey

Task	Main references
Maintaining Individual Influence	[31], [37], [38], [39], [40], [41], [42]
Maintaining Influential Seed Set	[42], [43], [44], [45], [46], [47], [48], [49]
Extracting Influential Vertices from Temporal Networks	[50], [51], [52], [53], [54], [55]
Probing Network Evolution for Influence Analysis	[56], [57], [58]
Learning Influence Weights	[59], [60], [61], [62], [63], [64], [65]

(followers and followees) and some of its actions (e.g., tweets and retweets). However, this information cannot directly tell us the up-to-date influence weights between this vertex and its neighbors. To learn the new influence weights, we need to use the data of users' recent actions, which is generated based on users' influence on each other.

This task can be combined with the probing task when we can only access limited user action data. The probing task only probes a few users' evolution in connections that are the most important to influence analysis in an evolving network. Thus, when we can only access to limited user action data, we can only crawl the user actions related to the users identified by the probing task and learn influence weights of edges incident to these users.

Worth noting that even for a network owner, learning the up-to-date influence weights is necessary because the network owner often only has user action data, and influence weights should be derived from user actions. Only with the up-to-date influence weights as input, the first three tasks of extracting influential users can be done effectively.

1.3 Relationship among the Major Tasks

The five tasks are related to each other. The first three tasks all aim at maintaining influential vertices in an evolving network. The difference among them is that each task adopts its own measure of influence. For the last two tasks, probing network evolution can guide the learning task, where we may only learn influence weights of edges incident to important vertices found by the probing task. The strategy in the probing task is guided by what influence analysis task we want to conduct in the evolving network. The probing stage and the learning stage together provide the input, which is the information (full or partial) of the network evolution, to the first three tasks. Fig. 1 shows the connections of the five tasks.

The rest of this survey is organized as follows. We briefly introduce essential concepts in influence models and evolving networks in Section 2. We introduce representative studies in the five major tasks in Sections 3, 4, 5, 6, and 7, respectively, which are listed in Table 1. In Section 3, we review representative studies in maintaining influential individuals in evolving networks. Algorithms for maintaining influential seed sets in evolving networks are introduced in Section 4. In Section 5, we elaborate the recent progress in incorporating network evolution into modeling influence for extracting influential vertices in temporal networks. Sections 6 and 7 are about how to detect the information of network evolution for influence analysis, where Section 6 focuses on deciding which vertices are important to influence diffusion in evolving networks and need to be monitored, while Section 7 discusses how to learn influence weights of certain edges. We conclude this survey and discuss some future directions in Section 8.

2 Preliminaries

In this section, we introduce the basic mathematical models to represent evolving networks and the most commonly used influence models for static networks.

We first define the notion of weighted graph, since both subjects are based on weighted graphs. A weighted graph is defined as $G = \langle V, E, w\rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and, for each edge $(u, v) \in E$, w_{uv} denotes its weight. Denote by $n = |V|$ the number of vertices. We use $N_{in}(u) = \{v \,|\, (v, u) \in E\}$ to indicate the set of u's in-neighbors, and $N_{out}(u) = \{v \,|\, (u, v) \in E\}$ to indicate the set of u's out-neighbors in G. Denote by $d_{in}(u) = \sum_{v \in N_{in}(u)} w_{vu}$ the in-degree of u and $d_{out}(u) = \sum_{v \in N_{out}(u)} w_{uv}$ the out-degree of u in G.

2.1 Evolving Networks

An evolving network is often modeled as a time series of static networks G^1, G^2, ..., G^t, ..., where $G^t = \langle V, E^t\rangle$[1] is called the snapshot at time t. Besides that, the update stream model is also often used to describe an evolving network. An update stream is represented by a time series of edge weight updates,[2] where an edge weight is represented by (u, v, Δ, t), which means the edge weight w_{uv} is updated to $w_{uv} + \Delta$ at time t. Modeling evolving networks as time series of snapshots is suitable for slowly-evolving networks, where two consecutive snapshots G^{t-1} and G^t normally do not differ much. Update streams are popularly adopted in studies of incremental/online algorithms on fast evolving networks [24], [26], [31], [32], [41], [66].

Examples of evolving networks include the web graph, online social networks, money transfer networks, etc. For

1. We assume the vertex set pre-exists and never changes.
2. Vertex insertion/deletion can be reduced to edge weight updates. We just need to set V large enough [24], [31], [41], [56], [64].

example, the retweeting network is an evolving network. In the retweeting network, if a user u retweets a message from a user v at time t, we update the network with with an edge weight update $(u, v, 1, t)$. In influence analysis, evolving networks are often built based on user action stream [54], [55], [64], [65], which indicates who performed what actions at what time. We will review how to employ user action stream to learn the edge weights in influence network in Section 7. User action stream is also an important data source in modeling users' temporal influence in a given time period, rather than at just a time point. We will deliberate that further in Section 5.

2.2 Influence Models for Static Networks

In influence analysis, the most fundamental problem is how to measure the influence of a vertex or a group of vertices. We introduce some popularly adopted influence/diffusion models for modeling influence in static networks in this section. Most studies in evolving network influence analysis are based on the influence models reviewed in this section.

2.2.1 Stochastic Diffusion Models

Stochastic diffusion models that model influence diffusion as a stochastic process taking place in the network are extensively studied. We introduce three most popular stochastic models in literature, the Linear Threshold (LT) model, the Independent Cascade (IC) model and the Continuous-Time Independent Cascade (CTIC) model. All the three models are based on well-known studies of the diffusion phenomenon in marketing research [67], [68], epidemiology [69], and behavior research in social science [70]. Most studies in influence maximization, one of the most important research problems in influence analysis, adopt these three models to measure influence of vertices [2], [3], [71], [72], [73], [74], [75].

For both the LT model and the IC model, given a network $G = \langle V, E, w\rangle$, the edge weight w_{uv} reflects the strength of u's influence on v.

Linear Threshold (LT) model. In the Linear Threshold model [2], each vertex $v \in V$ also carries a weight w_v, which is called the *self-weight* of v. Denote by $W_v = w_v + \sum_{u\in N_{in}(v)} w_{uv}$ the total weight of v. The *influence probability* of an edge (u, v) is

$$p_{uv} = \frac{w_{uv}}{W_v}.$$

Clearly, for $v \in V$, $\sum_{u\in N_{in}(v)} p_{uv} \leq 1$.

In the LT model, given a seed set $S \subseteq V$, the influence is propagated in G as follows. First, every vertex u randomly selects a threshold $\lambda_u \in [0, 1]$, which reflects our lack of knowledge about users' true thresholds. Then, influence propagates iteratively. Denote by S_i the set of vertices that are active in step i $(i = 0, 1, \ldots)$. Set $S_0 = S$. In each step $i \geq 1$, an inactive vertex v becomes active if

$$\sum_{u\in N_{in}(v)\cap S_{i-1}} p_{uv} \geq \lambda_v.$$

The propagation process terminates in step t if $S_t = S_{t-1}$.

Let $I(S)$ be the expected number of vertices that are finally active when the seed set is S. We call $I(S)$ the *influence spread* of S, which measures the influence of S. Let I_u be the influence spread of a single vertex u. Kempe et al. [2] proved that the LT model is equivalent to a "live-edge" process where each vertex v picks at most one incoming edge (u, v) with probability p_{uv}. Consequently, v does not pick any incoming edges with probability $1 - \sum_{u\in N_{in}(v)} p_{uv} = \frac{w_v}{W_v}$. All edges picked are "live" and the others are "dead". Then $I(S)$ is the expected number of vertices reachable from S through live edges.

Independent Cascade (IC) model. In the IC model [2], for an edge (u, v), $p_{uv} = w_{uv}$ is the *influence probability* $(0 \leq p_{uv} = w_{uv} \leq 1)$. Given a seed set S, the influence propagates in G in a way different from the LT model. Denote by S_i the set of vertices that are active in step i $(i = 0, 1, \ldots)$. Set $S_0 = S$. In each step $i \geq 1$, each vertex u that is newly activated in step $i - 1$ has a single chance to influence its inactive out-neighbor v with an independent probability p_{uv}. The propagation process terminates in step t when $S_t = S_{t-1}$.

Similar to the LT model, the influence spread $I(S)$ is the expected number of vertices that are finally active when the seed set is S. The equivalent "live-edge" process [2] of the IC model is to keep each edge (u, v) with a probability p_{uv} independently. All kept edges are "live" and the others are "dead". Then $I(S)$ is the expected number of vertices reachable from S via live edges.

Continuous-Time Independent Cascade Model. Rodriguez et al. [62] extended the IC model to the Continuous Time Independent Cascade Model, where the process of influence diffusion is continuous in time. In the CTIC model, each edge (u, v) is associated with a length distribution $f_{uv}(\tau; w_{uv})$, where w_{uv} is the parameter determining the strength of u's influence on v. Rodriguez et al. [62] discussed three types of the length distribution: $w_{uv}e^{-w_{uv}\tau}$ (Exponential), $\frac{w_{uv}}{\delta}(\frac{\tau}{\delta})^{-1-w_{uv}}$ (Power Law) and $w_{uv}\tau e^{-\frac{1}{2}w_{uv}\tau^2}$ (Rayleigh).

Given a seed set and a time threshold t, an influence propagation under the CTIC model first samples a length τ_{uv} for each edge (u, v), and then activates any vertex that can be reached from S via a path whose total length is no more than t. In the CTIC model, τ_{uv} indicates the time that u spends to influence v. Given a time threshold t, the influence spread of S is the expected number of vertices activated within time t in the influence propagation started from S.

Properties of Influence Spread. Influence spread under the IC model, the LT model and the CTIC model captures some intuitive properties of real-world influence diffusion. First, the influence spread $I(\cdot)$ is monotone [2], which means $I(S) \leq I(T)$ if $S \subseteq T$. Second, $I(\cdot)$ is submodular [2], also known as "diminishing return", which indicates that $I(S \cup \{u\}) - I(S) \geq I(T \cup \{u\}) - I(T)$ for any $S \subseteq T$. These two properties are used to design the greedy algorithm for influence maximization and prove its effectiveness [2].

Computing the influence spread of a given vertex set S is #P-hard for the IC model, the LT model and the CTIC model [71], [76], [77]. We often use Monte Carlo simulations [2], [71] or reverse Monte Carlo simulations (also called the polling method) [73], [74] to approximate influence spread.

In this survey, when discussing problems related to stochastic diffusion models, we focus on studies adopting the three models introduced above.

2.2.2 Alternative Influence Model: PageRank

As illustrated above, one shortcoming of stochastic diffusion models is that computing influence under these models is #P-hard. Thus, alternative influence models where influence of vertices is easy to compute are also adopted in literature. PageRank [78] is the most widely used alternative influence model or a heuristic baseline in influence analysis [2], [72], [76], [77], [79]. Thus, we also review studies on PageRank in evolving network in this survey.

Given a static network $G = \langle V, E, w\rangle$, the edge weight w_{uv} reflects how much u trust v or how strong v's influence on u. The probabilistic transition matrix A is an $n \times n$ matrix defined as follows.

$$A_{uv} = \begin{cases} 0 & \text{if } (u,v) \notin E \\ \frac{w_{uv}}{d_{out}(u)} & \text{if } (u,v) \in E \end{cases}. \quad (1)$$

PageRank is the stationary distribution of a random walk that, at each step, with a certain probability $1-\alpha$, jumps to a random vertex, and with probability α follows a randomly chosen outgoing edge from the current vertex. Specifically, the PageRank vector π is defined as the solution to the linear system

$$\pi = \alpha A\pi + (1-\alpha)\frac{\mathbf{1}}{n}, \quad (2)$$

where $\mathbf{1} = (1, 1, \ldots, 1)^\top$. The vth entry of π, π_v, can be regarded as the influence of vertex v in G. The PageRank vector can be efficiently computed by numerical methods such as the power iteration [80] and the Gauss-Southwell method [38].

Note that since π_v is a probability, it normally measures the importance or the relative influence (comparing to other vertices) of v but it cannot tell people the expected number of vertices that can be influenced by v. Moreover, PageRank often is only used to measure influence of individual vertices [14], [81], [82], [83], [84], rather than the influence of a group of vertices.

Graph Centrality Measures. PageRank is the most popular graph centrality measure. In some early investigations of influence analysis [2], [85], other graph centrality measures such as degree centrality and distance/closeness centrality were used as baselines. Below, we list some commonly used graph centrality measures.

- Degree centrality. The degree centrality of a vertex v is $|N_{out}(v)|$, which is the number of vertices following v in the network.
- Closeness/Distance centrality. In a fully connected graph, the closeness centrality of a vertex v is defined as $\frac{1}{\sum_{u\in V, u\neq v} d(v,u)}$, where $d(v,u)$ is the distance from v to u.
- Katz score. Given the adjacency matrix A of a graph, the Katz score of a vertex v is defined as $\sum_{u\in V}\sum_{i=1}^{\infty}\alpha^i(A^i)_{vu}$, where $\alpha \leq 1$ is a predefined decaying factor.
- Betweenness centrality. In a graph $G = \langle V, E\rangle$, where edges can be directed or undirected and may carry non-negative weights. Denote by $\mathcal{S}_{uv}$ the set of shortest paths from u to v and $\mathcal{S}_{uv}(w)$ the set of shortest paths from u to v passing w. The betweenness centrality of a vertex w is defined as $\frac{1}{|V|(|V|-1)}\sum_{(u,v)\in V\times V, u\neq v}\frac{|\mathcal{S}_{uv}(w)|}{|\mathcal{S}_{uv}|}$.

3 Maintaining Individual Influence against Network Evolutions

When we have the complete information of network evolution, which is a stream of edge weight updates, a critical problem is how we can update the influence of each individual vertex in an incremental manner. In this section, we first introduce two methods that efficiently update (approximate) PageRank values of vertices in an evolving network. We then discuss some studies in maintaining other graph centrality measures. We review the previous studies on how to maintain approximate influence spread of individual vertices under both IC and LT models against a stream of edge weight updates. Influential individuals can be extracted by drawing vertices with high influence (PageRank or influence spread under IC or LT models) maintained.

3.1 Maintaining PageRank Values

When a network updates a little bit, re-running the power iteration methods from scratch takes $O(\frac{m}{\epsilon})$ time to get a vector $\hat{\pi}$ such that $|\hat{\pi}_v - \pi_v| \leq \epsilon$ for all $v \in V$ [80]. Obviously, this naive method is too time-consuming considering that real networks are large (m, the number of edges, is large) and often evolve at a high rate. Thus, it is desirable to incrementally update PageRank values of vertices based on the update of network and the old PageRank values that we already have.

3.1.1 The Monte Carlo Method

One popular method for incremental maintenance of PageRank values is the Monte Carlo method proposed by Bahmani et al. [37]. The method is based on the probabilistic explanation of PageRank values [86], which is implied by another representation of the PageRank vector

$$\pi = \frac{1-\alpha}{n}(I-\alpha A)^{-1}\mathbf{1} = \left(\sum_{k=0}^{\infty}(\alpha A)^k\right)\frac{(1-\alpha)\mathbf{1}}{n}.$$

This equation indicates that we can draw some random walks starting from a random vertex to approximate PageRank values. For a random walk, at each step it terminates with probability $1-\alpha$, and makes a transition according to the matrix A with probability α. Suppose we draw R random walks starting from each vertex, and vertex v appears X_v times in these random walks. Bahmani et al. [37] pointed out that we can use $\hat{\pi}_v = \frac{(1-\alpha)X_v}{nR}$ to approximate π_v, because $\hat{\pi}_v$ is an unbiased estimation of π_v. To make $|\hat{\pi}_v - \pi_v| \leq \epsilon$ for all $v \in V$ with high probability, Bahmani et al. [37] further analyzed that setting $R = O(\ln n)$ is enough when the tolerable error ϵ is a constant.

The advantage of the Monte Carlo method [37] is that, when the network only changes a little bit, for example, there is an edge weight update (u, v, Δ, t), we do not have to re-generate all random walks. Instead, the random walks not passing u can still be used because they are not affected by the edge weight update. Only a few random walks passing u may need to be updated by rerouting from u. Bahmani et al. [37] proved that the cost of dealing with m random edge insertions is only $O(\frac{nR}{(1-\alpha)^2}\ln m)$, and the expected cost of dealing with a random edge deletion is

$O(\frac{nR}{m(1-\alpha)^2})$. These costs are much lower than $O(\frac{m}{\epsilon})$, the cost of rerunning the power iteration method for a single edge insertion/deletion, when ϵ is not too small.

One drawback of the Monte Carlo method [37] is that it needs $O(\frac{1}{\epsilon^2})$ random walks for achieving a tolerable error ϵ (assuming ϵ is a variable rather than a constant). If we require a highly accurate $\hat{\pi}$, for example we set $\epsilon = 10^{-6}$, $O(\frac{1}{\epsilon^2})$ is too large and the Monte Carlo method becomes not efficient in practice.

3.1.2 The Algebraic Method

To maintain highly accurate PageRank vector $\hat{\pi}$, Ohsaka et al. [38] proposed an algorithm based on Gauss-Southwell iterations [87]. Denote by $\hat{\pi}(i)$ the vector of $\hat{\pi}$ in the ith Gauss-Southwell iteration. Correspondingly, a residual vector $r(i)$ is defined as

$$r(i) = (1-\alpha)\frac{\mathbf{1}}{n} - (I - \alpha A)\hat{\pi}(i).$$

Obviously when $r(i)$ approaches 0, $\hat{\pi}(i)$ converges to the PageRank vector π defined in Eq. (2). In the ith iteration, the largest component (in absolute value) $r_j(i-1)$ in $r(i-1)$ is picked to update $\hat{\pi}(i-1)$ to $\hat{\pi}(i)$. Specifically,

$$\begin{aligned} &\hat{\pi}(i) \leftarrow \hat{\pi}(i-1) + r_j(i-1)e_j \\ &r(i) \leftarrow r(i-1) - r_j(i-1)e_j + \alpha r_j(i-1)Ae_j, \end{aligned} \tag{3}$$

where e_j is an n-dimensional vector with the jth entry as 1 and all other entries as 0's. If after the ith iteration, the largest component (in absolute value) $r_j(i)$ of $r(i)$ is no greater than ϵ, the Gauss-Southwell algorithm stops and we have $|\hat{\pi}_v(i) - \pi_v| \leq \epsilon$ for all $v \in V$. Note that an update according to Eq. (3) only needs $O(|N_{out}(j)|)$ time since only the residuals of vertices adjacent to j are affected.

To deal with updates of an evolving network, Ohsaka et al. [38] proposed to use $\hat{\pi}^{t-1}$, which is $\hat{\pi}$ maintained according to the Gauss-Southwell algorithm at time $t-1$, as the initial value $\hat{\pi}^t(0)$ of $\hat{\pi}^t$ for conducting Gauss-Southwell iterations at time t. Ohsaka et al. [38] proved that by doing so, when the network changes only one edge at time t, only $O(\frac{1}{\epsilon})$ iterations are needed to make $\hat{\pi}^t$ converge. Suppose the largest number of out-neighbors of vertices in the evolving network is no more than d. To handle an edge weight update, the method [38] only takes $O(\frac{d}{\epsilon})$ time. This is much better than the Monte Carlo method [37] which needs $O(\frac{1}{\epsilon^2})$ time to update random walks. Moreover, comparing to [37] where the space we need is $O(m + \frac{1}{\epsilon^2})$, [38] only needs $O(m)$ space.

3.2 Maintaining other Graph Centrality Measures

Besides PageRank, we also review some studies in maintaining other graph centrality measures such as closeness centrality, since a few early investigations of influence analysis [2], [85] used these measures as baselines.

The major idea of maintaining closeness/distance centrality against network updates is to detect the set of vertices that may be affected by an edge update. Sariyüce et al. [39] proposed a pruning method for edge insertion and deletion in an unweighted graph. Suppose the edge being inserted/deleted is (u, v). Sariyüce et al. [39] proved that if $|d(w,u) - d(w,v)| \leq 1$ for a vertex w, and $d(w,u)$ is the distance form w to u before the edge insertion/deletion, the closeness of w remains the same after the edge update. Kas et al. [88] devised an incremental algorithm based on the dynamic algorithm in [89] for updating closeness in dynamic weighted networks. Bisenius et al. [90] proposed to maintain top-k closeness centrality vertices in a dynamic network, where the idea is to detect the vertices whose closeness values are not affected by edge updates. Since the aim is to extract top-k vertices, Bisenius et al. [90] proposed to maintain approximations of vertices' closeness values to further accelerate the computation.

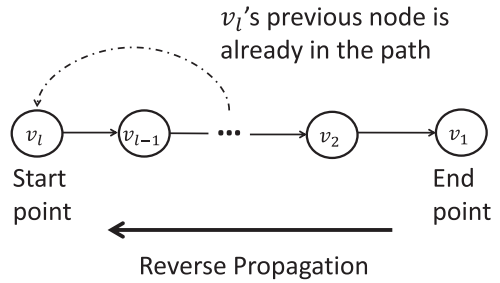
To maintain Katz scores of vertices in an evolving network, Nathan and Bader [40] exploited properties of iterative solvers for quickly updating approximate Katz scores when the underlying network is updated. Nathan et al. [91] showed that if we only want to detect the top-k Katz score vertices, we can reduce the approximation quality of the Katz scores maintained to reduce the computational cost. Grinten et al. [92] derived upper and lower bounds of vertices' Katz scores, and used these bounds to quickly update the vertex ranking based on Katz scores when the network is updated. Grinten et al. [92] also proposed a GPU implementation of their algorithm which can achieve a speedup of an order of magnitude.

The state-of-the-art evolving network betweenness centrality algorithms [31], [93] use a data structure called Hypergraph Sketch [31] to approximate betweenness scores of vertices, especially the top betweenness vertices. The Hypergraph Sketch is based on random sampling [31] and the quality of the approximate betweenness scores can be improved if we enlarge the size of the Hypergraph Sketch. Hayashi et al. [31] proposed an incremental algorithm to quickly update the Hypergraph Sketch. Riondato and Upfal [93] exploited Rademacher averages and pseudodimension, two fundamental concepts from statistical learning theory, to analyze the size of the Hypergraph Sketch and proposed an algorithm based on progressive random sampling to approximate betweenness scores of vertices in a dynamic graph. We do not review early progress in maintaining betweenness centrality in evolving networks because betweenness centrality was rarely used as a baseline in influence analysis [36]. Bonchi et al. [94] systematically reviewed recent progress in computing betweenness centrality in a dynamic graph and readers who are interested in this line of research can refer to the tutorial [94].

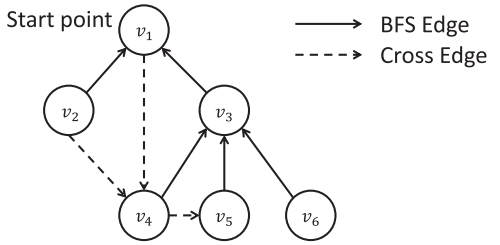
3.3 Maintaining Individual Influence Spread under Stochastic Diffusion Models

Unlike PageRank, computing individual influence under the LT and the IC models is #P-hard [76], [77]. Thus, maintaining accurate individual influence of vertices under the LT and the IC models in an evolving network probably is too costly to realize. At the same time, more often than not we only try to maintain individual influence of vertices up to a tolerable error, as long as the maintained individual influence can help us extract highly influential vertices with good qualities.

Yang et al. [41] proposed to incrementally update approximate individual influence of vertices against a stream of edge weight updates. The idea is to maintain a number of reverse reachable (RR for short) sets generated



(a) An RR set under the LT model is a random path.



(b) An RR set under the IC model is a random connect component.

Fig. 2. RR set.

by the polling method [73], and approximate individual influence of vertices by their frequencies in the RR sets maintained. To generate an RR set on a static graph G, we randomly pick a vertex u and run the "live edge" process of the LT or the IC model to find all vertices that can reach u via live edges. Then all vertices found form an RR set. Suppose we have an RR sketch composed of M RR sets $\{R_1, \ldots, R_M\}$. Denote by $\mathcal{D}(u)$ the number of RR sets containing u, or the degree of u in the RR sketch. Then we have that $\frac{n\mathcal{D}(u)}{M}$ is an unbiased estimation of I_u, the individual influence of u.

Suppose we already have RR sets $\{R_1, \ldots, R_M\}$ drawn from G^{t-1} and there is an edge weight update (u, v, Δ, t) on the last snapshot G^{t-1} at time t. Yang et al. [41] pointed out that, for both the LT model and the IC model, we do not need to regenerate another pile of RR sets but only need to update a few of the existing ones to make them equivalent to being generated from the up-to-date network G^t. How to update RR sets affected by an edge weight update is based on two critical observations of RR sets under the LT model and the IC model, which are shown in Fig. 2.

Similar to the Monte Carlo method for maintaining PageRank values [37], when an edge weight update (u, v, Δ, t) comes in, most existing RR sets that do not contain v do not need to be updated. In expectation only $\frac{MI_v^{t-1}}{n} \ll M$ RR sets need to be retrieved and may be updated to accommodate an edge update (u, v, Δ, t) [41], where M is the number of RR sets we already have and I_v^{t-1} is v's influence spread at time $t-1$. Empirical studies in [41] show that the incremental algorithm is much more efficient than re-generating all RR sets. For example, for the Twitter dataset [41], it takes more than 8000s to re-generate $M \approx 1.5$ million RR sets for a single edge update, while the incremental algorithm only takes less than 0.1ms to update the maintained RR sets for the same edge update.

Setting a proper sample size M is also an important issue. Unlike PageRank, where the PageRank vector π is a distribution so that $\sum_{v \in V} \pi_v = 1$, $\sum_{v \in V} \frac{I_v}{n}$ is not a constant. This property makes the sample size M not a constant in some situations. Thus, every time after updating RR sets against an edge weight update, we also need to check whether the current sample size M is proper. Since real networks are highly dynamic and edge weight updates arrive at a high rate, such a decision needs to be made efficiently. The decision should also help us maintain as few RR sets as possible. Moreover, due to the #P-hardness of computing influence spread, we often only maintain accurate approximate influence spread of influential individual vertices such that we can track influential vertices with quality guarantees. The sample size M can be determined according to specific tasks. Yang et al. [41], [42] studied two common tasks of tracking influential vertices in an evolving network, namely tracking vertices with influence spread of at least a threshold T and tracking top-k influential vertices.

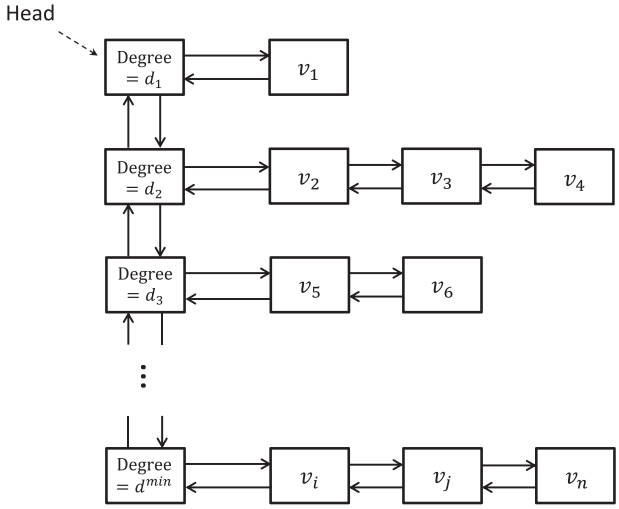


Fig. 3. The double linked list data structure.

For the thresholding task, given a threshold T, Yang et al. [41] proposed to set $M = \frac{12T}{n\epsilon^2} \ln \frac{2n}{\delta}$, and every time when a query of the threshold-based influential vertices is issued, return the set of vertices $S = \{u \,|\, n\frac{\mathcal{D}(u)}{M} \geq T - \frac{\epsilon n}{2}\}$. Yang et al. [41] proved that with probability at least $1 - \delta$, this simple algorithm can achieve that S contains all influential vertices, that is, recall being 100 percent, and $\min_{v \in S} I_v \geq T - \epsilon n$. Note that $M = \frac{12T}{n\epsilon^2} \ln \frac{2n}{\delta}$ is a constant that does not change with respect to edge weight updates.

For the top-k task, the sample size M may change as the network updates. Yang et al. [42] proposed to maintain the invariant $\mathcal{D}^k = \lceil \frac{4(e-2)(1+\epsilon)\ln 2n\delta}{\epsilon^2} \rceil$, where $\mathcal{D}^k$ is the kth largest value of $\mathcal{D}(v)$ for all $v \in V$. Thus, every time when $\mathcal{D}^k < \lceil \frac{4(e-2)(1+\epsilon)\ln(2n/\delta)}{\epsilon^2} \rceil$, we need to add extra RR sets into the RR sketch, and when $\mathcal{D}^k > \lceil \frac{4(e-2)(1+\epsilon)\ln(2n/\delta)}{\epsilon^2} \rceil$, we can delete some RR sets from the RR sketch. When a top-k query is issued, we return $S = \{v \,|\, \mathcal{D}(v) \geq \frac{1-\epsilon}{1+\epsilon}\mathcal{D}^k\}$. Yang et al. [42] proved that with probability at least $1 - \delta$, the recall of S is 100 percent, which means if $I_v \geq I^k$ (I^k is the kth greatesr individual influence spread) then $v \in S$, and $\min_{v \in S} I_v \geq (1 - \frac{4\epsilon}{1+\epsilon}) I^k$.

Yang et al. [42] also devised a double linked list data structure shown in Fig. 3, where the vertices with the same degree in the RR sketch are grouped in one same linked list

and the head nodes of all liked lists are sorted in the degree descending order. Using this data structure, we can always maintain vertices sorted by degrees in the RR sketch, and check if $\mathcal{D}^k = \lceil \frac{4(e-2)(1+\epsilon)\ln(2n/\delta)}{\epsilon^2} \rceil$ in $O(1)$ time.

3.4 Summary and Discussion

In this section, we review several incremental algorithms that maintain PageRank values [37], [38], graph centrality measures [31], [39], [40], [88], [90], [92], [93], and individual influence spreads [41], [42] of vertices in evolving networks.

PageRank, due to its probabilistic explanation and its algebraic closed form, can be maintained by both Monte Carlo methods and algebraic methods. The Gauss-Southwell iteration based algebraic method even can achieve updating cost $O(\frac{d}{\epsilon})$ and thus can maintain accurate PageRank values of vertices. However, as discussed in Section 2.2.2, the PageRank vector has some limitations in influence analysis since it cannot tell how many vertices can be influenced by a given vertex. Other popularly adopted graph centrality measures also have similar issues.

Influence spreads under both the IC and the LT models are #P-hard to compute. Thus, unlike PageRank, it is hard to derive efficient algebraic methods to maintain influence spreads. The RR sketch based methods [41], [42] can maintain top influential vertices well and the RR sketch only needs to be updated minorly when an edge weight update arrives. However, one shortcoming of the methods [41], [42] is that they have to deal with edge weight updates one by one. Designing a batch mode updating method for the RR sketch based methods is an open problem. Moreover, how to track influential vertices under the Continuous Time Independent Cascade model is still untouched.

4 Maintaining Influential Seed Set (for Influence Maximization) against Network Evolutions

Besides tracking influential individuals, influence maximization, which aims at finding a set of k vertices such that the combined influence is maximized, is also an important problem in influence analysis. Note that the combined influence of a set of vertices often does not equal the sum of influence spreads of all individuals in this set, because influence scopes of vertices often overlap.

In influence maximization, a set of k vertices used to trigger a diffusion of influence is called a k-seed set. Finding the optimal k-seed set is NP-hard [2]. Thus, the golden rule of influence maximization is to apply a greedy algorithm [2] to find a $(1 - 1/e - \epsilon)$ optimal seed set, where ϵ is the error of approximating influence spread [2], [72], [73], [74]. In the scenario of an evolving network, how to incrementally maintain a high quality k-seed set becomes a challenging research topic in influence analysis.

One big challenge in maintaining an influential seed set against network evolutions is maintaining influence spread of seed sets. The existing studies along this line can be categorized into three groups, maintaining proxies of influence spread, RR sketch based methods, and online algorithms. Note that a proxy of influence spread is not actually the real influence spread and the gap between these two normally does not have theoretical bounds, while for an RR sketch, as long as the number of RR sets is big enough, with high probabilities, from the RR sketch we can derive approximate influence spreads of vertices with small errors. Online algorithms take feedback from environment, that is, rewards/propagation results of the seed set picked by online algorithms, as input to adjust the strategy of selecting the next seed set for the future. Sometimes the edge updates are not needed in online algorithms.

4.1 Influence Spread Proxy Based Methods

Since computing the exact influence spread of a given seed set is #P-hard [76], [77], one heuristic way to estimate influence spread is to use proxies. Some existing studies utilize influence spread proxies and design efficient methods that exploit properties of the proxies adopted to efficiently update the maintained seed set when the network evolves.

4.1.1 SPIM Heuristic Proxy

One famous proxy of influence spread under the IC model is the SP1M heuristic [95]. Given a seed set S and a vertex v, the probability that v is influenced by S at step $t \leq d(S, v) + 1$ can be computed in $O(|E|)$ time [95], where $d(S, v)$ is the minimum number of hops from S to v. Based on this observation, SP1M only considers the probability of v being influenced by S within $d(S, v) + 1$ steps.

The UBI algorithm [43] adopts SP1M in maintaining an influential k-seed set S in an evolving network under the IC model. Chen et al. [43] modeled an evolving network as a sequence of snapshots G^1, G^2, ..., G^t. Based on the seed set S obtained from G^{t-1}, Chen et al. proposed an interchange greedy heuristic to replace vertices in S to get a new and good seed set when the network is updated to G^t. Denote by

$$\delta_{v,v_s}(S) = I(S \setminus \{v_s\} \cup \{v\}) - I(S),$$

the gain of replacing v_s by v. The SP1M heuristic is employed to estimate $I(S \setminus \{v_s\} \cup \{v\})$ and $I(S)$ to get an estimation of $\delta_{v,v_s}(S)$. In each iteration, UBI replaces v_s from S by $v \in V \setminus S$ such that the replacement gain

$$\delta_{v,v_s}(S) = \max_{v_1 \in S, v_2 \in V \setminus S} \delta_{v_1,v_2}(S).$$

UBI stops when $\delta_{v,v_s}(S) \leq \gamma I(S)$, where γ is a predefined small constant that controls the tradeoff between the efficiency and the effectiveness of the algorithm.

Just applying the SP1M heuristic to estimate $\delta_{v,v_s}(S)$ is still not scalable. To further improve the efficiency, Chen et al. [43] derived an upper bound of $\delta_{v,v_s}(S)$ using the upper bound of $I(S)$ discussed in [96]. This upper bound of $\delta_{v,v_s}(S)$ can be efficiently computed. When the network updates, values of $\delta_{v,v_s}(S)$ for all $v \in V \setminus S$ and $v_s \in S$ can be updated in $O(|E|)$ time. With the help of these upper bounds, when we need to pick v and v_s of the maximum interchange gain, many candidates of v and v_s can be pruned such that we do not need to run the SP1M heuristic to estimate their interchange gains. Song et al. [44] further proposed the UBI+ algorithm, an extension of UBI, where the major improvement is to derive an even tighter upper bound of $\delta_{v,v_s}(S)$ to achieve more effective pruning.

4.1.2 MIA Heuristic Proxy

Another proxy of influence spread under the IC model is the Maximum Influence Arborescence (MIA) heuristic [76],

whose idea is to only consider the local region of vertices to estimate influence spread. Given a seed set S, its influence spread $I(S)$ equals $\sum_{v \in V} \Pr(S, v, G)$, where $\Pr(S, v, G)$ is the probability that v is influenced by S. In MIA, when computing $\Pr(S, v, G)$, only maximum influence paths (MIPs) are considered. The maximum influence path from u to v is defined as

$$MIP(u, v, G) = \arg \max_{\rho \in \mathcal{P}(u,v,G)} \Pr(\rho), \tag{4}$$

where $\mathcal{P}(u, v, G)$ is the set of all paths from u to v in G, and for a path $\rho = \{u_1, u_2, \ldots, u_\tau\}$, $\Pr(\rho) = \prod_{i=1}^{\tau-1} p_{u_i u_{i+1}}$ is its diffusion probability. MIA also adopts a parameter θ to filter out MIPs whose diffusion probabilities are smaller than θ. Thus, in MIA, if θ is set properly, $\Pr(S, v, G)$ can be calculated efficiently by only considering a few vertices that can influence u via MIPs.

Liu et al. [45] proposed the IncInf algorithm, which exploits the MIA heuristic as a proxy of influence spread to maintain an influential k-seed set in an evolving social network. Based on the results of running the greedy algorithm for influence maximization [85] on each snapshot of real evolving networks, Liu et al. [45] observed that, when an evolving network is updated, the vertices of the seed set produced by the greedy algorithm usually are those having high degrees or whose individual influence spreads are increased substantially. Thus, when the network is updated, instead of running the greedy algorithm on all vertices, IncInf [45] runs the greedy algorithm on the vertices that have high degrees or whose influence spreads are increased substantially due to the network update.

The MIA heuristic helps accelerate the greedy algorithm [76] and detect vertices whose influence spreads change substantially. The intuition is that the impact of an update on an edge is local, since the MIA heuristic only considers the local regions of vertices in computing influence spread. For each vertex $u \in V$, we define two local regions of u, the Maximum Influence Out-Arborescence (MIOA) and the Maximum Influence In-Arborescence (MIIA) as follows.

$$MIOA(u, \theta, G) = \cup_{v \in V, \Pr(MIP(u,v,G)) \geq \theta} MIP(u, v, G) \tag{5}$$

$$MIIA(u, \theta, G) = \cup_{v \in V, \Pr(MIP(v,u,G)) \geq \theta} MIP(v, u, G), \tag{6}$$

where θ is the predefined parameter to filter out unimportant MIPs, $MIOA(u, \theta, G)$ is the local region that u can influence, and $MIIA(u, \theta, G)$ is the local region that u can be influenced by. $MIOA$ and $MIIA$ are used in the MIA-based greedy algorithm [76]. To update $MIOA(u, \theta, G)$ and $MIOA(u, \theta, G)$ for each $u \in V$ and detect the vertices whose influence spreads are increased much, Liu et al. [45] found that only the influence spreads of vertices that appear in $MIIA(u, \theta)$ may be affected. Thus, only for those vertices we need to compute their changes in influence spread and their $MIOA$.

However, IncInf is still not scalable since even running the greedy algorithm [76] on a small set of potential seeds is costly. When adopting MIA to compute influence spread, the influential k-seed set S is produced by running the MIA-based greedy algorithm [76], where in the ith iteration, the vertex u_i with the largest marginal gain $MG(S_{i-1}, u_i) = I(S_{i-1} \cup \{u_i\}) - I(S_{i-1})$ is added to the current seed set $S = S_{i-1}$ (with $i-1$ seeds) to update S to $S_i = S_{i-1} \cup \{u_i\}$ (with i seeds). Based on this observation, Yalavarthi et al. [46] proposed the N-family method to quickly detect the largest k' such that the first k' seeds in the current seed set S are still be picked as the first k' seeds if we run the greedy algorithm [76] on the current snapshot of the network. After finding k', the N-family method only needs to run the greedy algorithm [76] to collect the rest $k - k'$ seeds to finish updating the k-seed set S maintained. The N-family method is about 1$\sim$3 orders of magnitude faster than the UBI+ algorithm [43] in the experiments of [46].

4.2 RR Sketch Based Methods

As illustrated in Section 3.3, an RR sketch that contains M RR sets can approximate real influence spreads well, no matter the diffusion model is the IC model or the LT model, as long as the sample size M is big enough.

In all RR sketch based influence maximization algorithms on static networks [72], [73], [74], [97], most of the running time is spent on generating enough RR sets. Running the greedy algorithm [2] on the RR sketch to get a seed set, which is called an influence maximization query, only takes very little time. Thus, by utilizing the RR sketch in evolving network influence maximization, our goal is to maintain a RR sketch such that with high probability (at least $1 - \frac{1}{n}$), by running the greedy algorithm [2] on the sketch we can obtain a $(1 - \frac{1}{e} - \epsilon)$-optimal seed set with respect to the up-to-date snapshot of the evolving network.

Both [47] and [41] discussed how to update only a few existing RR sets against network updates under the IC model. Yang et al. [41] also devised an algorithm to update the RR sketch under the LT model. The aim is to keep the RR sketch an approximation of influence spreads of vertices in the up-to-date snapshot of the evolving network. So every time when the network is updated, we first apply the method in [47] and [41] to update the RR sketch maintained. After that, one important step is to decide if the current sample size M, i.e., the number of RR sets in the RR sketch, is proper. Note that in the environment of an evolving network, the proper sample size M may vary as the network evolves. If we find the sample size M is too small, we add some new RR sets to the RR sketch. If we find M is too large, we delete some RR sets from the RR sketch. Since real networks evolve rapidly, it is desirable that deciding if the sample size M is proper can be done efficiently. This makes the methods deciding the sample size M in some influence maximization algorithms for static networks [72], [74], [97] not applicable to evolving networks, because their complexity is $O(\sum_{i=1}^{M} |R_i|)$ where R_i is the ith RR set in the sketch and $|R_i|$ is its number of vertices.

$\mathcal{C}(\mathcal{R})$, the cost of the RR sketch $\mathcal{R}$, can be used to decide if the current sample size M is proper [73]. The cost of a RR set R_i containing vertices $u_1, \ldots, u_\kappa$ is defined as $\mathcal{C}(R_i) = \kappa + \sum_{i=1}^{\kappa} |N^{in}(u_i)|$ where $N^{in}(u_i)$ is the set of in-neighbors of u_i in the up-to-date snapshot. The cost of an RR sketch $\mathcal{R} = \{R_1, \ldots, R_M\}$ is $\mathcal{C}(\mathcal{R}) = \sum_{i=1}^{M} \mathcal{C}(R_i)$. Since we can always record the cost of an RR set, $\mathcal{C}(\mathcal{R})$ can be retrieved in $O(1)$ time.

To maintain a proper sample size M against a stream of network updates, Ohsaka et al. [47] proposed to maintain the invariant $\mathcal{C}(\mathcal{R}) = \Omega(\frac{(m+n)\log n}{\epsilon^3})$ to get a $(1 - \frac{1}{e} - \epsilon)$-optimal

seed set with high probability, where m is the number of edges of the network and n is the number of vertices. However, $\Omega(\frac{(m+n)\log n}{\epsilon^3})$ is very large in practice, Ohsaka et al. [47] only maintained the invariant $\mathcal{C}(\mathcal{R}) = 32(m+n)\log n$ in their experiments, which is far from the theoretical value $\Omega(\frac{(m+n)\log n}{\epsilon^3})$. What's more, $\Omega(\frac{(m+n)\log n}{\epsilon^3})$ is not a correct bound since according to the latest version of [73] this bound should be $\Omega(\frac{k(m+n)\log n}{\epsilon^2})$ where k is the size of seed set.

Yang et al. [42] proposed to use $\mathcal{D}^* = \arg\max_{u \in V} \mathcal{D}(u)$ as the signal to decide if the current sample size M is proper, where $\mathcal{D}(u)$ is the number of RR sets containing u. By the linked list data structure in [41], retrieving $\mathcal{D}^*$ takes $O(1)$ time. Yang et al. [42] analyzed that when an influence maximization query is issued, to return $(1-\frac{1}{e}-\epsilon)$-optimal k-seed set with probability at least $1-\frac{1}{n}$, we should maintain the invariant that $\mathcal{D}^* = \lceil \Upsilon_1(\frac{\epsilon}{2-1/e}, \frac{2}{3n\binom{n}{k}}) \rceil$ where $\Upsilon_1(\epsilon, \delta) = 1 + (1+\epsilon)\frac{4(e-2)\ln\frac{2}{\delta}}{\epsilon^2}$. Yang et al. [42] further derived that their method maintains an RR sketch with $O(\frac{kn\ln n}{I^*\epsilon^2})$ RR sets, where $I^* = \arg\max_{u \in V} I_u$ is the largest individual influence spread. Moreover, $E[\mathcal{C}(\mathcal{R})]$, the expected cost of the RR sketch maintained by [42], is upper bounded by $O(\frac{k(m+n)\ln n}{\epsilon^2})$ and in practice $E[\mathcal{C}(\mathcal{R})]$ is much smaller than $\Omega(\frac{k(m+n)\ln n}{\epsilon^2})$. Thus, in practice the expected cost of the RR sketch maintained by [42] usually is much smaller than the cost reported in [47].

However, $O(\frac{kn\ln n}{I^*\epsilon^2})$ RR sets are still too many in practice, especially when k is big. Yang et al. [42] proposed a practical solution that maintains the invariant $\mathcal{D}^* = \frac{1}{2}\lceil \Upsilon_1(\frac{\epsilon}{2-1/e}, \frac{2}{3n^2}) \rceil$, which is similar to [47] where we maintain only $\frac{1}{k}$ times of the correct number of RR sets. Extensive experiments in [42] show that the practical solution of [42] maintains much fewer RR sets and is 1 or 2 orders of magnitude faster than [47], , while the quality of seed sets extracted by [42] is not compromised.

Yang et al. [42] also devised a highly efficient implementation of the greedy algorithm [2] for influence maximization queries run on the RR sketch maintained. The major idea is to exploit the linked list data structure [41] that maintains an approximate rank of vertices based on their influence spreads, and use the submodularity of the influence spread function to prune most vertices in the process of greedily adding seed vertices. The experiments of [42] showed that this efficient implementation is an order of magnitude faster than than the implementation of the greedy algorithm of [47].

4.3 Online Algorithms

Online algorithms often do not require the edge updates as the input but only require feedbacks of influence diffusions from the environment. Different from the methods introduced above, which aim at finding a good seed set for the current network, online algorithms seek for seeding strategies that in the long run (for example, for T snapshots of an evolving network) the total influence spread of selected seed sets is good.

Bao et al. [48] proposed the problem of online influence maximization in evolving networks. Denote by G^t the snapshot of the evolving network at time t, and $f_t(S)$ the number of users influenced by S in G^t. Note that $f_t(S)$ is a random variable and its expectation is $I_t(S)$, the influence spread of S in G^t. Suppose the total time span is T. Let S^* be the offline optimal k-seed set, that is, $S^* = \arg\max_{S \subseteq V, |S|=k} \sum_{t=1}^{T} I_t(S)$. Let $OPT = \sum_{t=1}^{T} I_t(S)$. The goal of [48] is to pick a k-seed set S_t for each snapshot G^t, such that the greedy weak regret

$$Reg_G(T) = \left(1 - \frac{1}{e}\right) OPT - \sum_{t=1}^{T} E[I_t(S_t)],$$

is minimized.[3] Bao et al. [48] devised a bandit algorithm which does not require the structure of the evolving network but only queries the value of $f_t(S)$. In the bandit algorithm [48], each vertex is assigned with an importance weight, which is adjusted every time when receiving the value of $f_t(S)$ for the seed set S that we want to query. The bandit algorithm [48] selects a vertex u as the next seed to add to S with probability proportional to u's current importance weight. Bao et al. [48] proved that their bandit algorithm has a greedy weak regret $Reg_G(T) = O(\sqrt{Tn\log n})$.

Wu et al. [49] proposed the Evolving Influence Maximization (EIM) problem for fast-evolving networks under the IC model. The motivation is that during influence diffusion, the network may evolve a lot. Thus, when given the current snapshot G^t, instead of selecting a seed set S_t to maximize influence spread in G^t, Wu et al. [49] proposed to select S_t for maximizing influence spread in G^{t+1}, the next snapshot. A bandit-based framework $\mathbb{EIM}$ was devised for the EIM problem. In each step t, $\mathbb{EIM}$ first predicts the structure of G^{t+1} based on the current snapshot G^t and observed newly added vertices, then learns the evolving influence weights of edges from previous influence diffusion feedbacks, and selects S_t by applying influence maximization algorithms. Define the scaled regret of $\{S_1, S_2, \ldots, S_T\}$, the output of $\mathbb{EIM}$, as

$$Reg(T) = \sum_{t=1}^{T} [I_{t+1}(S_t^*) - \frac{1}{\beta} I_{t+1}(S_t)],$$

where β is the approximation ratio of influence maximization algorithms [72], [74] for the IC model and $S_t^* = \arg\max_{S \subseteq V, |S|=k} I_{t+1}(S)$. Wu et al. [49] proved that if the number of edges in G^{T+1} is m, then the scaled regret of $\mathbb{EIM}$ is $O(\sqrt{mT\ln T})$.

4.4 Summary and Discussion

In this section, we review the incremental algorithms to maintain an influential k-seed set [42], [43], [44], [45], [46], [47], [48], [49] in an evolving network. Influence spread proxy based methods (heuristic methods) [43], [44], [45], [46], RR sketch based methods [42], [47], and online/bandit algorithms [48], [49] are reviewed.

The major drawback of the influence spread proxy based methods [43], [44], [45], [46] is that they cannot compute influence spreads of vertices with theoretical guarantees so the quality of the results may be poor. Moreover, these heuristic methods [43], [44], [45], [46] all have parameters that need to be carefully set to fit specific datasets. [43], [44] have

3. The reason of using $(1-\frac{1}{e})OPT$ in the above equation is that computing OPT is NP-hard and by applying the greedy algorithm [2] we can get an $(1-\frac{1}{e})$-approximation of OPT.

the parameter γ to make the interchange heuristic converge. [45], [46] need to set the parameter θ carefully, otherwise either many important MIPs may be filtered out (when θ is set too large) or the computation of estimating influence spread based on the MIA heuristic may be too slow (when θ is set too small).

For the RR sketch based methods [43], [44], [45], [46], the core problem is how to efficiently decide a proper and tight sample size M. Both [42], [47] adopt very efficient decisions that can be done in $O(1)$ time to decide if the current sample size M is proper. However, even the more efficient one [42] between these two methods needs to maintain $O(\frac{kn\ln n}{I^*\epsilon^2})$ RR sets, which are still too many in practice when k is large. Compared to $O(\frac{kn\ln n}{I_k^*\epsilon^2})$,[4] the number of RR sets generated by influence maximization algorithms for static networks [72], [74], [97], the sample size $M = O(\frac{kn\ln n}{I^*\epsilon^2})$ still has rooms to be improved. However, as illustrated above, decisions of sample sizes in [72], [74], [97] are too costly in time and are not applicable to networks evolving rapidly. Thus, devising efficient decision methods on sample size for maintaining an influential k-seed set with big k is still an open problem.

The major idea of online algorithms [48], [49] is adjusting importance weights of vertices upon receiving feedbacks of previous diffusions. One drawback of $\mathbb{EIM}$ [49] is that it requires edge-level feedback, that is, whether a specific edge is activated in a diffusion. However, in reality, what we usually have is vertex-level feedback, which means we often only know who are influenced but we do not know who is the influencer of an influenced user [98], [99]. A potential improvement for both [48] and [49] is designing incremental seed set selection algorithms. For a new snapshot, both the two online algorithms [48], [49] need to re-do seed set selection from scratch, which does not utilize the past results and is not efficient.

5 Extracting Influential Vertices from Temporal Networks

In the previous two sections, the influence spread we discussed is defined on the up-to-date snapshot of an evolving network. However, since the up-to-date snapshot is still a static graph without the dimension of time, the influence spread discussed so far is also static. In some evolving social networks such as epidemiological networks, edges may be rapidly added or deleted. In such a case, traditional influence spread defined on a static snapshot may not be meaningful for some applications because a diffusion itself takes time and during this time the network structure may change substantially.

In this section, we review some important works in modeling the time dimension in influence analysis and extracting influential vertices with high temporal influence spreads. Unlike traditional influence analysis where the influence diffuses in a static network, temporal influence models assume that a diffusion process takes place in an evolving network. The aim is to mine vertices that are influential during a time period that involves multiple snapshots of the evolving network.

4. I_k^* is the influence spread of the optimal k-seed set. Note that $I^* \le I_k^* \le kI^*$.

5.1 Dynamic Interaction Based Methods

Aggarwal et al. [51] proposed to use the frequently evolving user interactions to model temporal influence of users. To calculate the influence of a set of vertices in a time-period (t_0, t_0+h) in an evolving network $G = (G^1, G^2, \ldots, G^t, \ldots)$, they first split the time interval (t_0, t_0+h) into time periods $(t_0, t_1, \ldots, t_r = t_0+h)$. Denote by $\pi(u,t)$ the probability that u is influenced at time t. Then the temporal influence of S in the time period (t_0, t_0+h) is

$$I(S, t_0, t_0+h) = \sum_{u\in V} \pi(u, t_0+h). \tag{7}$$

Aggarwal et al. [51] proposed a method to calculate $\pi(u,t)$ step by step from time t_0. Specifically, given a seed set S, Aggarwal et al. [51] set $\pi(u,t_0) = 1$ if $u \in S$ and $\pi(u,t_0) = 0$ otherwise. Given two adjacent time points t_{τ_1} and t_{τ_2} in the time periods $(t_0, t_1, \ldots, t_r = t_0+h)$, suppose we know $\pi(u, t_{\tau_1})$ for every $u \in V$. The method [51] calculates $\pi(u, t_{\tau_2})$ by

$$\pi(u,t_{\tau_2}) = \pi(u,t_{\tau_1}) + [1-\pi(u,t_{\tau_1})]\Big\{1 - \prod_{v\in IN(u,t_{\tau_1},t_{\tau_2})} [1-\pi(v,t_{\tau_1})p_{vu}(t_{\tau_1},t_{\tau_2})]\Big\}, \tag{8}$$

where $IN(u, t_{\tau_1}, t_{\tau_2})$ is the set of all vertices interacting with u in the time period (t_{τ_1}, t_{τ_2}) and $p_{vu}(t_{\tau_1}, t_{\tau_2})$ is the probability that v influences u during (t_{τ_1}, t_{τ_2}). Suppose the edge (v,u) exists multiple times during (t_{τ_1}, t_{τ_2}) and the lengths of its existences are $\delta t_1, \delta t_2, \ldots, \delta t_r$. Aggarwal et al. [51] defined $p_{vu}(t_{\tau_1}, t_{\tau_2})$ as $1 - \prod_{i=1}^{r}[1 - f_{vu}(\delta t_i)]$ where $0 \le f_{vu}(\delta t_i) \le 1$ can be plugged into any function monotonic to δt_i.

Based on the above definition of temporal influence, Aggarwal et al. [51] devised a Forward Influence algorithm to find the most influential set of vertices in time period (t_0, t_0+h). Aggarwal et al. [51] also devised a Backward Influence algorithm to detect the seed vertices at time t_0 given the diffusion result at time t_0+h.

5.2 Temporal Diffusion Model Based Methods

Adapting widely used diffusion models to fit evolving networks is a potential solution to model temporal influence of vertices in literature.

PageRank can be represented by

$$\pi_u = \sum_{v\in V}\sum_{k=0}(1-\alpha)\alpha^k \sum_{z\in Z(v,u),|z|=k} \Pr(z),$$

where $Z(v,u)$ is the set of all walks from v to u and $\Pr(z)$ is the probability of generating a walk z. Motivated by this random walk explanation, Rozenshtein et al. [50] proposed Temporal PageRank for an evolving network $G = \langle V, E\rangle$, where $E = \{(u_i, v_i, t_i) \,|\, i = 1, 2, \ldots, m\}$. (u_i, v_i, t_i) represents at time t_i there is an interaction between u_i and v_i. In [50], a temporal walk is defined as a sequence of edges (u_1, u_2, t_1), $(u_2, u_3, t_2), \ldots, (u_j, u_{j+1}, t_j)$ such that $t_i \le t_{i+1}$ for all $1 \le i \le j-1$. Let $Z(v, u \,|\, t)$ be the set of all temporal walks from v to u before time t. Rozenshtein et al. [50] proposed how to compute $\Pr(z \,|\, t)$, which is the probability of generating a temporal walk z before time t. Then the temporal PageRank value of u at time t is

$$r_u(t) = \sum_{v \in V} \sum_{k=0}^{t} (1-\alpha)\alpha^k \sum_{z \in Z(v,u\,|\,t), |z|=k} \Pr(z\,|\,t).$$

Rozenshtein et al. [50] devised a streaming algorithm to compute temporal PageRank values by only scanning all edges once.

Gayraud et al. [52] modified the two most popular diffusion models, the IC model and the LT model, to model how diffusions of influence take place in an evolving network. The idea is to have the evolution time and the diffusion time run in lock-step. At time point t, the snapshot G^t is locked when a diffusion step of the IC model or the LT model happens, and then the network updates from G^t to G^{t+1} and we enter time point $t+1$. This idea is general enough to include the possibility that the diffusion time runs faster than the evolution time. If s diffusion steps are executed on a snapshot G^i, we can simply simulate this process by adding s copies of the snapshot G^i, and again have the evolution time and the diffusion time run in lock-step. Similarly, if diffusion is slower than evolution, we can aggregate the multiple snapshots that correspond to a single diffusion step.

Another issue in modeling temporal influence is to determine when a newly influence vertex is able to influence its neighbors. Gayraud et al. [52] considered two situations, namely transient influence and persistent influence. In transient influence, if a vertex u is influenced at time t, u can only try to influence its neighbors at time $t+1$. In persistent influence, if u is influenced, it has a chance to influence another vertex v at the first time when v follows u.

Gayraud et al. [52] proved that if we assume transient influence, for both the IC model and the LT model, the modified models which simulate diffusions in evolving networks as described above have non-submodular influence spread functions. For the IC model under such a situation the influence spread function is even not monotone. When we assume persistent influence, if the edge weight p_{uv} stays fixed for every edge (u, v) existing in multiple snapshots of the evolving network G, then the modified models are both monotone and submodular [52].

Xie et al. [53] adapted the continuous time influence model [100], which is a generalization of the IC model, to the evolving network setting. Instead of assuming that we have all snapshots of an evolving network [52], Xie et al. [53] assumed that we have a base network G^0 and the evolution of G in the future is random and can be described by a set of parameters $f_1, \ldots, f_s$. A diffusion process starting from a given seed set S is modeled by a Continuous Time Markov Chain (CTMC) [101]. Xie et al. [53] proved that the influence spread function is monotone and submodular under this new model. Thus, we can maximize influence spread using the greedy algorithm [2]. However, computing influence spread under the model in [53] is very hard because the Continuous Time Markov Chain may have an exponential number of states.

5.3 Information Flow Based Methods

Comparing to building a temporal influence model, a more direct way to extract temporarily influential vertices is to use a database approach, which is to query the user interacting action log to find who influenced the biggest number of users in the given period. In social networks like Twitter and Facebook, interacting actions of a user like retweeting and replying can tell us who influenced this user. Such actions can be regarded as directed edges and form an evolving network since user actions all have timestamps. We can trace back user interacting actions to find the origin user of a specific action. For example, we may trace back retweet records to find who posted this tweet at the first place. In literature, a path from the origin user to a user influenced via interacting actions is often called an *information flow*.

Wang et al. [54] defined an action as $a_t = \langle u, a_{t'} \rangle_t$ $(t' < t)$, where u is the user who performs this action, t is the timestamp and $a_{t'}$ denotes an action performed at time $t' < t$ that influences u to take the action a_t. If u performs an original action at time t, such as posting an original tweet, this action is denoted by $a_t = \langle u, nil \rangle$. u influences v in the time period (t_a, t_b) if we can find an information flow $(a_{t_1} = \langle u_1 = u, a_{t'} \rangle_{t_1}, a_{t_2} = \langle u_2, a_{t_1} \rangle_{t_2}, a_{t_3} = \langle u_3, a_{t_2} \rangle_{t_3}, \ldots, a_{t_\tau} = \langle u_\tau = v, a_{t_{\tau-1}} \rangle_{t_\tau})$ where $t_a \le t_1 < t_2 \cdots < t_\tau \le t_b$. Since we often care about the most recent user actions, the query time period (t_a, t_b) is often set as the past period where there are t actions. Denote by $(u \rightsquigarrow v)_t$ if u influenced v in the past period with t actions. Then

$$I_t(S) = \{v \,|\, v \in V, (S \rightsquigarrow v)_t\}, \tag{9}$$

is the temporal influence of S during the past period, where S is a seed set and V is the set of all users. Since user actions come as a stream, Wang et al. [54] devised the Sparse Influential Checkpoints (SIC) framework to maintain an influential seed set S that has high temporal influence in the past period against the user action stream. The SIC method selectively keeps $O(\frac{\log t}{\beta})$ checkpoints and the seed set S maintained is $\frac{\epsilon(1-\beta)}{2}$optimal.

Subbian et al. [55] proposed to use keywords in user postings and the social network structure to determine who influences whom. Denote by $\mathcal{Q}$ the set of all keywords appearing in user postings. A valid flow of a keyword K_i from a user u to another user v is defined as a user sequence $\mathcal{P} = (u = u_1, u_2, \ldots, u_\tau = v)$ where u_{i+1} posts something with the keyword K_i after u_i has a posting with keyword K_i and (u_i, u_{i+1}) is an edge in the social network, for all $i = 1, 2, \ldots, \tau - 1$. The influence of u on v at the current time t is defined as

$$I(u, v, \mathcal{Q}, t) = \sum_{K_i \in \mathcal{Q}} \sum_{\mathcal{P} \in S_{uv}} \mathcal{A}(\mathcal{P}, K_i, t),$$

where S_{uv} is the set of all possible paths from u to v and $\mathcal{A}(\mathcal{P}, K_i, t)$ is a scoring function for a path $\mathcal{P}$ with keyword K_i. The score of $\mathcal{P}$ depends on its start time $t_\mathcal{P}$ and the current time t that reflects that the influence defined is temporal, that is sensitive to time. Then the temporal influence of a user u is defined as

$$\alpha(u, \mathcal{Q}, t) = \frac{\sum_{v \in V} I(u, v, \mathcal{Q}, t)}{\sum_{u_1, u_2 \in V} I(u_1, u_2, \mathcal{Q}, t)}.$$

Subbian et al. [55] demonstrated that this influence measure had close relationship to Katz centrality. Subbian et al. [55]

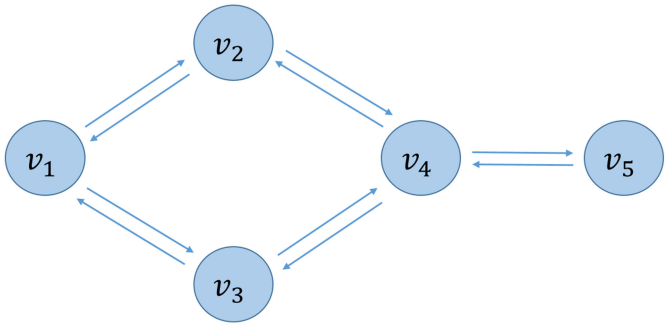


Fig. 4. An example of causality in influence diffusion.

also devised efficient algorithms to maintain user temporal influence $\alpha(u, \mathcal{Q}, t)$ against a stream of user postings, where the idea was to utilize a tree-like structure to maintain all valid flows.

5.4 Summary and Discussion

In this section, we reviewed the representative studies in modeling temporarily influential vertices in evolving networks. The methods based on dynamic interactions [51], temporal diffusion models [50], [52], [53] and information flows [54], [55] are revisited.

The major deficiency of the dynamic interaction based method defined [51] in Eq. (8) is that it violates the causality in influence diffusion. Taking Fig. 4 as an example, suppose v_1 is the only seed. The second part of Eq. (8) implies that the probability that v_4 is influenced depends on such probabilities of all its neighbors. Thus, based on Eq. (8), $\pi(v_4, t_{\tau_2})$ depends on $\pi(v_5, t_{\tau_1})$. However, it is clear that when v_1 is the only seed, only after v_4 is influenced can v_5 be influenced and $\pi(v_4, t_{\tau_2})$ should not depend on $\pi(v_5, t_{\tau_1})$.

The three temporal diffusion models [52], [53] elegantly incorporate network evolutions into modeling diffusions of influence. Open problems of these models are majorly in computation. For the temporal IC model and the temporal LT model [52], when vertices can only try to influence their neighbors right after they are influenced, the influence spread function is no longer submodular, and sometimes even not monotone. How to design efficient and effective influence maximization algorithms under such situations remains a challenge. The Continuous Time Markov Chain model [101] may have an exponential number of states and thus computing influence spread under this model is a challenging problem.

The information flow [54], [55] based methods reviewed in this section directly mine temporarily influential users from user action data. The drawback is that they require substantial user interaction data, while in real social networks, many user actions like sharing and posting may not indicate by whom the user is influenced to perform an action. Thus, in reality we may have substantial user actions in the form $a_t = \langle u, nil \rangle_t$ such that the definition of influence in Eq. (9) may suffer an issue of data sparsity. On the contrary, [55] makes a too loose assumption of who influences whom, where if v follows u and v posts something with a word appearing in the tweets posted by v before, u is an influencer of v. The common drawback of [54], [55] is that they all tend to assign a high influence value to a user who performs many actions, even though this user's actions usually can only get a small number of other users influenced.

6 Probing Network Evolution for Influence Analysis

For a real evolving social network, more often than not, except for the social network owner, no one has the full access to the changes of the network. If one wants to conduct influence analysis on the social network, she normally needs to call APIs of the network to access changes of some of the vertices in the network. Thus, one important problem is to decide which vertices to probe in order to conduct influence analysis with good qualities.

6.1 Probing for PageRank Computation

Bahmani et al. [56] studied the problem of probing the connections of only one vertex at each time point for PageRank calculation. The goal is to make H^t, the network snapshot observed at time t, similar to G^t, the real snapshot at time t, in terms of PageRank values of vertices. Denote by π^t the real PageRank vector at time t, and ϕ^t the PageRank vector estimated on the observed snapshot H^t. To evaluate the effectiveness of ϕ^t given π^t, Bahmani et al. [56] adopted the L_∞ metric and the L_1 metric, which are defined as

$$L_\infty(\pi^t, \phi^t) = \max_{u \in V} |\pi_u^t - \phi_u^t|,$$

and

$$L_1(\pi^t, \phi^t) = \sum_{u \in V} |\pi_u^t - \phi_u^t|,$$

respectively.

Four probing algorithms, namely Random Probing, Round-Robin Probing, Proportional Probing and Priority Probing are proposed [56]. Random probing every time selects a uniformly random vertex and probes the neighbors of the vertex, while Round-Robin probing de-randomizes this process by cycling through the vertices and probing them in this order. Proportional probing selects u at time t to probe with probability proportional to ϕ_u^t, the PageRank value of u at time t estimated by the observed snapshot H^t. Like Round-Robin probing de-randomizing Random probing, Priority Probing also de-randomizes Proportional Probing to make sure that each vertex u is probed at least once every $O(1/\phi_u^t)$ steps. Specifically, Priority Probing maintains a priority value for each vertex, which is 0 at the beginning. In each time point t, we probe u that has the highest priority score and have the observed snapshot H^t. We calculate ϕ^t on H^t. Then we set u's priority to 0 and set the priority of v to ϕ_v^t for all $v \neq u$.

Experiments on real and synthetic evolving networks show that Priority Probing is always the best (with the lowest L_∞ and L_1 values) and Random Probing is always the worst (with the highest L_∞ and L_1 values). Moreover, the de-randomized probing methods are always better than their corresponding randomized probing methods. Bahmani et al. [56] further assumed a stylized yet arguably natural model of graph evolution. In this model, in every time point we randomly pick an edge and change the vertex that it points to to a new vertex. The new vertex is selected with probability proportional to the current PageRank value of the vertex,

which intuitively captures the importance of this vertex at that time. Under this model, Bahmani et al. [56] gave a theoretical analysis why Proportional Probing and Priority Probing outperform Random Probing and Round-Robin Probing.

6.2 Probing for Influence Maximization

Influence maximization is one of the most important tasks in influence analysis. Facing highly dynamic social networks, how to probe network evolution has a great impact on the quality of the seed set mined by influence maximization. To make influence maximization effective, the goal of probing is to make the quality of the seed set $\hat{S}$ mined from H^t, the snapshot observed at time t, not much worse than the seed set S mined from G^t, the real snapshot at time t.

Zhuang et al. [57] assumed that an evolving network changes smoothly, specifically, in each time point for each vertex u, the out-degree $d_{out}(u)$ changes at most by 1. Moreover, denote by $d^t_{out}(u)$ the out-degree of u at time t and $E[d^{t+1}_{out}(u) \mid d^t_{out}(u)]$ the conditional expectation of $d^{t+1}_{out}(u)$ given $d^t_{out}(u)$, Zhuang et al. [57] assumed $E[d^{t+1}_{out}(u) \mid d^t_{out}(u)] = d^t_{out}(u)$ which means $d_{out}(u)$ is relatively stable. Denote by H^t_v the observed snapshot obtained by probing v. Given a seed set S, Zhuang et al. [57] adopted $\hat{Q}_v(S)$, the degree centrality or the total out-degree of S in H^t_v, as a proxy of the influence spread of S in H^t_v. Let S be the set of k vertices that have the highest out-degrees in H^{t-1}, the observed snapshot at time $t-1$. Denote by S_v the seed set with k vertices that have the highest out-degrees in H^t_v. Under the above assumptions of network evolution, Zhuang et al. [57] analyzed that, by probing a vertex v in a time point t, given a parameter ϵ, we can figure out a deviation $\beta(v)$ such that $\Pr\{\hat{Q}_v(S_v) - \hat{Q}_v(S) \geq \beta(v)\} \leq \epsilon$. Thus, Zhuang et al. [57] devised a Maximum Gap Probing method, where given the parameter ϵ, in each time point the vertex v with the largest corresponding deviation $\beta(v)$ is probed. If in each time point we can probe multiple vertices, say b vertices, the Maximum Gap Probing method iteratively picks b vertices with the maximum deviations.

Han et al. [58] relaxed the assumptions of network evolution in [57] by assuming the total out-degree of vertices in a whole community is relatively stable. Based on this assumption, Han et al. [58] proposed to partition all vertices in a network into communities and devised a method to probe a vertex in each community for the purpose of influence maximization. The analysis of [58] is similar to [57].

6.3 Summary and Discussion

In this section, we review the methods probing an evolving network for influence analysis tasks including PageRank computation [56] and influence maximization [57], [58]. All these methods have explicit or implicit assumptions that the evolving network is relatively stable in the number of edges. This means these methods are not applicable to fast evolving networks where the number of edges keep growing.

Moreover, all the probing methods discussed are for observing a snapshot that is similar to the real snapshot in graph structure, since they all treat edges as unweighted and just use degrees of vertices to do probing. However, in influence analysis, influence weights of edges are really important and treating them as equal may lead to poor results in mining influential users [102]. Thus, in the next section, we discuss how to learn influence weights.

7 Learning Influence Weights

To collect enough information to conduct influence analysis, we need to learn influence weights on newly inserted or existing edges in the evolving network.

To learn influence weights, we need the log of user actions, which can tell us which vertex performs what action at what time. Most works in learning influence weights assume that the network where diffusion of influence happens is static. However, we can still apply them in learning influence weights of evolving networks, since real social networks evolve smoothly and we can regard an evolving network as stable in a time period $[t-h, t]$ as long as we keep h small enough. Then we can apply an influence weight learning method for static networks to learn from an evolving graph in the period $[t-h, t]$ using users actions in $[t-h, t]$. The output of these influence weight learning methods is the input of the methods in Sections 3, 4, and 5. Thus, we focus on the methods for learning influence weights in static networks.

One nice property of influence weight learning methods is that the computation can be decomposed into n independent parts, where n is the number of vertices, and each part corresponds to learning weights of in-edges of each vertex. Thus, influence weights of edges pointing to different vertices can be efficiently learnt in parallel. Based on this nice property, we can also utilize the output of the probing methods reviewed in Section 6 to reduce computation. We can apply probing methods first and then only learn influence weights of edges incident to vertices that are detected as important by methods reviewed in Section 6.

7.1 Maximum Likelihood Based Methods

Saito et al. [59] were the first to derive the likelihood of a propagation to learn influence weights of a static network under the IC model. A propagation is represented by a sequence of sets of users $D(t_a), D(t_a+1), \ldots, D(t_b)$ where t_a and t_b are the starting time point and ending time point of the propagation. $D(t)$ denotes the set of users influenced in the propagation at time t, which can be extracted from the user action log. Based on the IC model, the probability that a user u is influenced in a propagation at time t is

$$P_u(t) = 1 - \prod_{v \in N_{in}(u)} (1 - p_{vu}). \tag{10}$$

Denote by $C(t) = \cup_{i=t_a}^{t} D(i)$ the set of users that are influenced no later than time t. Then the likelihood of the propagation is

$$\begin{aligned} L(\boldsymbol{p}) = &\left(\prod_{t=t_a}^{t_b-1} \prod_{u \in D(t+1)} P_u(t+1) \right) \\ &\times \left(\prod_{t=t_a}^{t_b-1} \prod_{u \in D(t)} \prod_{v \in N_{out}(u) \setminus C(t+1)} (1 - p_{uv}) \right), \end{aligned} \tag{11}$$

where $\boldsymbol{p} = \{p_{uv}\}$ is the set of influence weights that we want to learn. Note that the second part of the righthand side of

Eq. (11) is the probability that the influenced users fail to influence their out-neighbors at the next time point right after they are influenced. Saito et al. [59] devised an EM algorithm to maximize $L(\boldsymbol{p})$ to learn influence weights. However, the EM algorithm cannot guarantee to converge to the optimal solution to maximize $L(\boldsymbol{p})$ in Eq. (11).

Note that Eq. (10) has an implicit assumption that we know the network structure since both the in-neighbor set $N_{in}(u)$ and the out-neighbor set $N_{out}(u)$ are used. There are also works [60], [61] that do not assume a known network structure and instead try to learn the structure, that is, the in-neighbor set of each vertex.

Netrapalli et al. [60] formulated the log-likelihood of a propagation under the IC model as a concave function. A new assumption made in [60] is that the seed set of a propagation is random where each vertex u becomes a seed with probability p_{init}. Let $t_u = t$ be the time when u is influenced in the propagation ($u \in D(t)$). Denote by $\theta_{uv} \equiv -\log(1 - p_{uv})$, the log-likelihood of a propagation $D(t_a), D(t_a + 1), \ldots, D(t_b)$ with s seeds can be written as

$$L(\boldsymbol{\theta}) = \log\left(p_{init}^{s}(1 - p_{init})^{n-s}\right) + \sum_{u \in V} L_u(\theta_{*u}), \tag{12}$$

where $\theta_{*u} = \{\theta_{vu} \mid v \in V\}$ and $L_u(\theta_{*u})$ is defined as

$$L_u(\theta_{*u}) = -\sum_{t=t_a}^{t_u-2} \sum_{v \in D(t)} \theta_{vu} + \log\left(1 - \exp\Big(-\sum_{v \in D(t_u-1)} \theta_{vu}\Big)\right). \tag{13}$$

The second part of the righthand side of Eq. (12) is actually taking log of Eq. (11).

From the above equations we can see that the problem of learning influence weights can be decomposed into independent subproblems where each subproblem aims at learning influence weights of edges pointing to a vertex. The objective of a subproblem to the vertex u is Eq. (13), which is concave so we can find the maximum. Assume that we have some prior knowledge of the in-neighbors of u, for example, we know that the set of u's in-neighbors is a subset of the set $\mathcal{S}_u$ and has a cardinality d_u. Netrapalli et al. [60] analyzed that, under such an assumption and if we have $M = O(d_u^2 \log \frac{|\mathcal{S}_u|}{\delta})$ random propagation records, with probability $1 - \delta$, we can find all in-neighbors of u that have strong influence weight on u.

Pouget et al. [61] further reduced the number of random propagation records needed to $M = O(d_u \log |\mathcal{S}_u|)$, under certain assumptions of the Hessian matrix $\nabla^2 L_u(\theta_{*u})$. The idea is to add a sparse regularization term $-\lambda|\theta_{*u}|_1$ to Eq. (13). Pouget et al. [61] proved that by doing so not only the in-neighbor set of u can be recovered with quality guarantees, but also p_{vu} can be recovered up to an additive error term.

Gomez-Rodriguez et al. [62] derived the log-likelihood of a propagation under the CTIC model. A propagation record is represented by a vector $\mathbf{t}^c = (t_1^c, \ldots, t_n^c)$, where t_u^c denotes the time when u got influenced in the propagation c. If u is not influenced in c, $t_u^c = \infty$. Given the length distribution $f_{uv}(\tau; w_{uv})$ for each $u, v \in V$, denote by

$$S(t_v^c \mid t_u^c; w_{uv}) = 1 - \int_0^{t_v^c - t_u^c} f_{uv}(\tau; w_{uv}) d\tau,$$

the probability that v is still not influenced by u (influenced at time t_u^c) at time t_v^c. Then the probability that v is influenced by u at time t_u^c is $f_{uv}(t_v^c - t_u^c; w_{uv}) \prod_{t_k < t_v, k \neq u} S(t_v^c \mid t_k^c; w_{kv})$. Denote by $P_v(\mathbf{t}^c; w)$ the probability that v gets influenced at time t_v^c given the influence weight $w = \{w_u v \mid (u, v) \in E\}$. If $t_v^c < \infty$, we have

$$P_v(\mathbf{t}^c; w) = \sum_{t_u < t_v} f_{uv}(t_v^c - t_u^c; w_{uv}) \prod_{t_k < t_v, k \neq u} S(t_v^c \mid t_k^c; w_{kv}).$$

If $t_v^c = \infty$ which means v is not influenced in the propagation c, then

$$P_v(\mathbf{t}^c; w) = \prod_{t_k < t_v} S(t_v^c \mid t_k^c; w_{kv}).$$

Thus, the log-likelihood of a propagation $\mathbf{t}^c = (t_1^c, \ldots, t_n^c)$ is $f(\mathbf{t}^c; w) = \sum_{v \in V} \log P_v(\mathbf{t}^c; w)$, and the log-likelihood of C, a collection of propagations, is

$$\sum_{c \in C} f(\mathbf{t}^c; w) = \sum_{c \in C} \sum_{v \in V} \log P_v(\mathbf{t}^c; w). \tag{14}$$

We can apply convex optimization algorithms to maximize Eq. (14) because it is a concave function [62]. Gomez-Rodriguez et al. [103] devised a stochastic gradient descent algorithm to maximize $\sum_{c \in C} w_c * f(\mathbf{t}^c; w)$, which is a weighted sum of log-likelihoods of propagations. By giving old propagations small weights, we can penalize the importance of old propagations.

Similar to [59], [60], since $P_v(\mathbf{t}^c; w)$ only involves $w_{*v} = \{w_{uv} \mid u \in V\}$, maximizing $f(\mathbf{t}^c; w)$ can also be decomposed into independent subproblems of learning $w_{*v} = \{w_{uv} \mid u \in V\}$ for each $v \in V$. Daneshmand et al. [104] proved that by adding an l_1-regularizer to Eq. (14), the network structure can be recovered with high probability if we have $O(d^3 \log n)$ propagation records, where d is the maximum in-degree of all vertices.

7.2 Other Methods

Some studies along this line only aim at learning the structure of a diffusion network from propagation data. Gomez-Rodriguez et al. [105] proposed to use maximum spanning tree and submodular function optimization algorithms to infer the edge set of a diffusion network. Tahani et al. [106] proposed a Hidden Markov model (HMM) to learn the edge set of a time-evolving diffusion network $G = \langle V, (E^1, E^2, \ldots, E^K)\rangle$ where E^k is the edge set of G at the kth time interval. Neither [105] nor [106] can provide us influence weights of edges for influence analysis.

When we have user interaction data which indicates the influencer of each user action, such as retweets, an intuitive idea to assign up-to-date influence weights to edges is the Probabilistic Edge Decay (PED) model [63]. Xie et al. [63] proposed to set the edge weight of e as $f(t - t(e)) = p^{t-t(e)}$, where t is the current time and $t(e)$ is the time when the edge e is inserted into the evolving network. The PED model considers both continuity and recency of users' interactions. One drawback of [63] is that when we want to learn influence weights of edges, the data we have usually only indicates

who performed what actions at what time but does not contain the information of the influencer of an action.

Goyal et al. [64] proposed three heuristics to assign an influence weight to an edge in an evolving network $G = \langle V, E, \mathcal{T} \rangle$, where $\mathcal{T} : E \rightarrow N$ is a function and $\mathcal{T}(u, v)$ indicates the time when the edge (u, v) is created. Denote by A_u the number of actions performed by u, $A_{u\&v}$ the number of actions performed by both u and v, and $A_{u|v} = A_u + A_v - A_{u\&v}$ the number of actions performed by u or v. Let A_{u2v} be the number of actions propagated from u to v. We say an action a to be propagated from u to v if: (1) v performs a at time $t_v(a)$ and u performs a at time $t_u(a) < t_v(a)$; (2) $(u, v) \in E$ and $\mathcal{T}(u, v) \leq t_u(a)$. The three heuristics [64] to assign influence weights work are as follows.

- *Bernoulli distribution.* For an edge (u, v), the influence weight p_{uv} is set to $\frac{A_{u2v}}{A_u}$.
- *Jaccard Index.* Set $p_{uv} = \frac{A_{u2v}}{A_{u|v}}$.
- *Partial Credits (PC).* For each action a performed by v at time $t_v(a)$, suppose S is the set of v's in-neighbors such that $\forall w \in S$, $\mathcal{T}(u, v) \leq t_u(a) < t_v(a)$. Define $credit_{u,v}(a)$ the credit of the edge (u, v) on action a. Let $credit_{u,v}(a) = \frac{1}{|S|}$ if $S \neq \emptyset$ and $credit_{u,v}(a) = 0$ otherwise. Then there are two instances of the PC methods.
  1) *Bernoulli model with partial credit.* Set $p_{uv} = \frac{\sum_a credit_{u,v}(a)}{A_u}$.
  2) *Jacard model with partial credit.* Set $p_{uv} = \frac{\sum_a credit_{u,v}(a)}{A_{u|v}}$.

Note that none of these methods can learn influence weights with guarantees under the popularly adopted diffusion models. Although the three methods are all heuristics, they are easy to compute. Goyal et al. [64] further extended these three heuristics by considering the time difference $t_v - t_u$ where t_u is the time u performed an action and t_v is the time v performed the same action. Goyal et al. [64] devised an efficient algorithm that only needs to scan the log of user actions twice to learn influence weights based on the heuristics proposed. Kutzhov et al. [65] proposed a fully stream learning algorithm that provides an approximation of the influence weights learnt by the Jaccard Index method.

7.3 Summary and Discussion

In this section, we review the studies [59], [60], [61], [62], [64], [65] in learning influence weights of edges, which are used as input to the evolving network influence analysis tasks illustrated in Sections 3, 4, and 5. We reviewed principled works that maximize the likelihood of propagation records [59], [60], [61] and heuristic methods [63], [64], [65] that assign influence weights to edges fast.

There are many challenges unsettled yet along this line. For example, similar to [65] that provides streaming algorithms to the heuristic methods in [64], can we design streaming algorithms that take a stream of user actions as the input to maximize the likelihood function defined in [59], [60], [61], [62]? Moreover, if we only care about edges with heavy influence weights, which can be regarded as the most important edges of the diffusion network, can we devise even faster algorithms (compared to [59], [60], [61], [62], [64], [65]) to just detect these important edges?

8 Conclusions and Future Directions

In this survey, we conduct an extensive review on influence analysis in evolving networks. We categorize the research in this topic into five major tasks, where the first three tasks focus on extracting influential vertices from an evolving network when the network evolution is completely known, and the other two tasks detect the network evolution for effective influence analysis when the network evolution is not fully accessible. We review representative works in the five major tasks. We also discuss some future research directions. To the best of our knowledge, this is the first survey on influence analysis in evolving networks, which is one of the most important research fields in influence analysis since almost all real networks where influence diffusion takes place are highly dynamic.

Besides the five major tasks discussed in this survey, there are also some other research challenges and directions in evolving network influence analysis. We list some of them below.

- *Influence analysis with very limited information about the network evolution.* In Sections 6 and 7 we discuss when we do not have the complete information about network evolution, we can probe and learn. However, for many social networks, for example, an epidemiological network, it is very hard to detect its structure let alone learning the influence weights on edges. Thus, we need to study how to conduct influence analysis in an evolving network where we only have very limited information of the evolution. Designing online algorithms like [48], [49] may be a potential solution but still faces many challenges. For example, the results of influence diffusions from selected seed sets may be only partially accessible to us, which makes [48], [49] not applicable.
- *Complex influence analysis tasks in evolving networks.* Sections 3, 4, and 5 are conducting influence analysis tasks when the network evolution is known. However, these three tasks are all just extracting influential vertices. In applications of influence analysis, for example, viral marketing, there are many other aspects we need to consider. For example, we need to consider how to spend some money to motivate influential users in a social network to spread their influence for us, under a limited budget [3]. We also need to consider the marketing effect after the influence diffusion is done, which means not just the number of users influenced is important, but also who are influenced [107]. To tackle these problems we need more complex algorithms. Conducting these complex influence analysis tasks in an evolving network is an interesting direction to explore.
- *Deep learning for evolving network influence analysis.* Recently, a deep neural network model [108] demonstrates its superiority in predicting social influence diffusion over traditional influence models. However, the deep learning model [108] is for static networks. Given the high accuracy of the deep learning model [108] for predicting influence diffusion in static networks, one possible future direction is to design deep learning methods for modeling influence

diffusion in evolving networks. Some factors considered by traditional generative models like the interplay between network evolution and influence diffusion [109] can guide us design effective deep learning models for evolving networks.

REFERENCES

[1] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 57–66.

[2] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 137–146.

[3] Y. Yang, X. Mao, J. Pei, and X. He, "Continuous influence maximization: What discounts should we offer to social network users?" in *Proc. Int. Conf. Manage. Data*, 2016, pp. 727–741.

[4] C. Long, R. C.-W. Wong, and V. J. Wei, "Profit maximization with sufficient customer satisfactions," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 2, 2018, Art. no. 19.

[5] Y. Pan, F. Cong, K. Chen, and Y. Yu, "Diffusion-aware personalized social update recommendation," in *Proc. 7th ACM Conf. Recommender Syst.*, 2013, pp. 69–76.

[6] I. Alina Christensen and S. Schiaffino, "Social influence in group recommender systems," *Online Inf. Rev.*, vol. 38, no. 4, pp. 524–542, 2014.

[7] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila, "Finding effectors in social networks," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 1059–1068.

[8] B. A. Prakash, J. Vreeken, and C. Faloutsos, "Spotting culprits in epidemics: How many and which ones?" in *Proc. IEEE 12th Int. Conf. Data Mining*, 2012, pp. 11–20.

[9] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 420–429.

[10] W. Xie, F. Zhu, J. Xiao, and J. Wang, "Social network monitoring for bursty cascade detection," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 4, 2018, Art. no. 40.

[11] A. Goyal, et al., "Discovering leaders from community actions," in *Proc. 17th ACM Conf. Inf. Knowl. Manage.*, 2008, pp. 499–508.

[12] D. Saez-Trumper, G. Comarela, V. Almeida, R. Baeza-Yates, and F. Benevenuto, "Finding trendsetters in information networks," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1014–1022.

[13] E. B. Khalil, et al., "Scalable diffusion-aware optimization of network topology," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1226–1235.

[14] G. Loukides and R. Gwadera, "Preventing the diffusion of information to vulnerable users while preserving PageRank," *Int. J. Data Sci. Analytics*, vol. 5, pp. 19–39, 2018.

[15] J. Weng, et al., "TwitterRank: Finding topic-sensitive influential twitterers," in *Proc. 3rd ACM Int. Conf. Web Search Data Mining*, 2010, pp. 261–270.

[16] N. Agarwal, et al., "Identifying the influential bloggers in a community," in *Proc. Int. Conf. Web Search Data Mining*, 2008, pp. 207–218.

[17] C. Chen, H. Tong, B. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. Chau, "Node immunization on large graphs: Theory and algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 113–126, Jan. 2016.

[18] R. Yan, et al., "Rumor blocking through online link deletion on social networks," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 2, 2019, Art. no. 16.

[19] X. Wei, N. C. Valler, B. A. Prakash, I. Neamtiu, M. Faloutsos, and C. Faloutsos, "Competing memes propagation on networks: A network science perspective," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 6, pp. 1049–1060, Jun. 2013.

[20] B. A. Prakash, D. Chakrabarti, N. C. Valler, M. Faloutsos, and C. Faloutsos, "Threshold conditions for arbitrary cascade models on arbitrary networks," *Knowl. Inf. Syst.*, vol. 33, no. 3, pp. 549–575, 2012.

[21] J. Leskovec, et al., "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2005, pp. 177–187.

[22] J. Leskovec, et al., "Microscopic evolution of social networks," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 462–470.

[23] A. Angel, et al., "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *Proc. VLDB Endowment*, vol. 5, no. 6, pp. 574–585, 2012.

[24] B. Bahmani, et al., "Densest subgraph in streaming and MapReduce," *Proc. VLDB Endowment*, vol. 5, no. 5, pp. 454–465, 2012.

[25] A. Epasto, et al., "Efficient densest subgraph computation in evolving graphs," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 300–310.

[26] S. Bhattacharya, et al., "Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams," in *Proc. 47th Annu. ACM Symp. Theory Comput.*, 2015, pp. 173–182.

[27] J. J.-C. Ying, et al., "FrauDetector+: An incremental graph-mining approach for efficient fraudulent phone call detection," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 6, 2018, Art. no. 68.

[28] T. L. Fond, J. Neville, and B. Gallagher, "Designing size consistent statistics for accurate anomaly detection in dynamic networks," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 4, 2018, Art. no. 46.

[29] H. Wang, J. Wu, W. Hu, and X. Wu, "Detecting and assessing anomalous evolutionary behaviors of nodes in evolving social networks," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 1, 2019, Art. no. 12.

[30] M. Nasre, M. Pontecorvi, and V. Ramachandran, "Betweenness centrality–incremental and faster," in *Proc. Int. Symp. Math. Found. Comput. Sci.*, 2014, pp. 577–588.

[31] T. Hayashi, et al., "Fully dynamic betweenness centrality maintenance on massive networks," *Proc. VLDB Endowment*, vol. 9, no. 2, pp. 48–59, 2015.

[32] N. Kourtellis, G. D. F. Morales, and F. Bonchi, "Scalable online betweenness centrality in evolving graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 9, pp. 2494–2506, Sep. 2015.

[33] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Comput. Surv.*, vol. 47, no. 1, 2014, Art. no. 10.

[34] J. Sun and J. Tang, "A survey of models and algorithms for social influence analysis," in *Social Network Data Analytics*. Berlin, Germany: Springer, 2011, pp. 177–214.

[35] A. Guille, H. Hacid, C. Favre, and D. A. Zighed, "Information diffusion in online social networks: A survey," *ACM SIGMOD Rec.*, vol. 42, no. 2, pp. 17–28, 2013.

[36] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 10, pp. 1852–1872, Oct. 2018.

[37] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized PageRank," *Proc. VLDB Endowment*, vol. 4, no. 3, pp. 173–184, 2010.

[38] N. Ohsaka, T. Maehara, and K.-I. Kawarabayashi, "Efficient PageRank tracking in evolving networks," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 875–884.

[39] A. E. Sariyüce, K. Kaya, E. Saule, and Ü. V. Çatalyiirek, "Incremental algorithms for closeness centrality," in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 487–492.

[40] E. Nathan and D. A. Bader, "A dynamic algorithm for updating katz centrality in graphs," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2017, pp. 149–154.

[41] Y. Yang, Z. Wang, J. Pei, and E. Chen, "Tracking influential individuals in dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2615–2628, Nov. 2017.

[42] Y. Yang, Z. Wang, T. Jin, J. Pei, and E. Chen, "Tracking top-K influential vertices in dynamic networks," *arXiv: 1803.01499*, 2018.

[43] X. Chen, G. Song, X. He, and K. Xie, "On influential nodes tracking in dynamic social networks," in *Proc. SIAM Int. Conf. Data Mining*, 2015, pp. 613–621.

[44] G. Song, Y. Li, X. Chen, X. He, and J. Tang, "Influential node tracking on dynamic social network: An interchange greedy approach," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 2, pp. 359–372, Feb. 2017.

[45] X. Liu, et al., "On the shoulders of giants: Incremental influence maximization in evolving social networks," *Complexity*, vol. 2017, 2017, Art. no. 5049836.

[46] V. K. Yalavarthi and A. Khan, "Fast influence maximization in dynamic graphs: A local updating approach," *arXiv: 1802.00574*, 2018.

[47] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-I. Kawarabayashi, "Dynamic influence analysis in evolving networks," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1077–1088, 2016.

[48] Y. Bao, X. Wang, Z. Wang, C. Wu, and F. C. Lau, "Online influence maximization in non-stationary social networks," in *Proc. IEEE/ACM 24th Int. Symp. Quality Serv.*, 2016, pp. 1–6.

[49] X. Wu, L. Fu, J. Meng, and X. Wang, "Evolving influence maximization," *arXiv: 1804.00802*, 2018.

[50] P. Rozenshtein and A. Gionis, "Temporal PageRank," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2016, pp. 674–689.

[51] C. C. Aggarwal, S. Lin, and P. S. Yu, "On influential node discovery in dynamic social networks," in *Proc. SIAM Int. Conf. Data Mining*, 2012, pp. 636–647.

[52] N. T. Gayraud, E. Pitoura, and P. Tsaparas, "Diffusion maximization in evolving social networks," in *Proc. ACM Conf. Online Social Netw.*, 2015, pp. 125–135.

[53] M. Xie, Q. Yang, Q. Wang, G. Cong, and G. De Melo, "DynaDiffuse: A dynamic diffusion model for continuous time constrained influence maximization," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 346–352.

[54] Y. Wang, Q. Fan, Y. Li, and K.-L. Tan, "Real-time influence maximization on dynamic social streams," *Proc. VLDB Endowment*, vol. 10, no. 7, pp. 805–816, 2017.

[55] K. Subbian, C. C. Aggarwal, and J. Srivastava, "Querying and tracking influencers in social streams," in *Proc. 9th ACM Int. Conf. Web Search Data Mining*, 2016, pp. 493–502.

[56] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal, "PageRank on an evolving graph," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 24–32.

[57] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun, "Influence maximization in dynamic social networks," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2013, pp. 1313–1318.

[58] M. Han, M. Yan, Z. Cai, Y. Li, X. Cai, and J. Yu, "Influence maximization by probing partial communities in dynamic online social networks," *Trans. Emerging Telecommun. Technol.*, vol. 28, no. 4, 2017, Art. no. e3054.

[59] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *Proc. Int. Conf. Knowl.-based Intell. Inf. Eng. Syst.*, 2008, pp. 67–75.

[60] P. Netrapalli and S. Sanghavi, "Learning the graph of epidemic cascades," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 211–222, 2012.

[61] J. Pouget-Abadie and T. Horel, "Inferring graphs from cascades: A sparse recovery framework," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 977–986.

[62] M. G. Rodriguez, D. Balduzzi, and B. Schölkopf, "Uncovering the temporal dynamics of diffusion networks," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 561–568.

[63] W. Xie, Y. Tian, Y. Sismanis, A. Balmin, and P. J. Haas, "Dynamic interaction graphs with probabilistic edge decay," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 1143–1154.

[64] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *Proc. 3rd ACM Int. Conf. Web Search Data Mining*, 2010, pp. 241–250.

[65] K. Kutzkov, A. Bifet, F. Bonchi, and A. Gionis, "STRIP: Stream learning of influence probabilities," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 275–283.

[66] M.-J. Lee, J. Lee, J. Y. Park, R. H. Choi, and C.-W. Chung, "QUBE: A quick algorithm for updating betweenness centrality," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 351–360.

[67] J. Goldenberg, B. Libai, and E. Muller, "Talk of the network: A complex systems look at the underlying process of word-of-mouth," *Marketing Lett.*, vol. 12, no. 3, pp. 211–223, 2001.

[68] 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 Marketing Sci. Rev.*, vol. 2001, 2001, Art. no. 1.

[69] R. M. Anderson, R. M. May, and B. Anderson, *Infectious Diseases of Humans: Dynamics and Control*, vol. 28. Hoboken, NJ, USA: Wiley Online Library, 1992.

[70] M. Granovetter, "Threshold models of collective behavior," *Amer. J. Sociology*, vol. 83, no. 6, pp. 1420–1443, 1978.

[71] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha, "Scalable influence estimation in continuous-time diffusion networks," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3147–3155.

[72] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 75–86.

[73] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proc. 25th Annu. ACM-SIAM Symp. Discr. Algorithms*, 2014, pp. 946–957.

[74] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1539–1554.

[75] B. Lucier, et al., "Influence at scale: Distributed computation of complex contagion in networks," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 735–744.

[76] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 1029–1038.

[77] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *Proc. IEEE 10th Int. Conf. Data Mining*, 2010, pp. 88–97.

[78] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," in *Proc. 7th Int. World Wide Web Conf.*, 1998, pp. 161–172.

[79] Q. Liu, B. Xiang, N. J. Yuan, E. Chen, H. Xiong, Y. Zheng, and Y. Yang, "An influence propagation view of PageRank," *ACM Trans. Knowl. Discovery Data*, vol. 11, no. 3, 2017, Art. no. 30.

[80] A. N. Langville and C. D. Meyer, "Deeper inside PageRank," *Internet Math.*, vol. 1, no. 3, pp. 335–380, 2004.

[81] N. Ma, J. Guan, and Y. Zhao, "Bringing PageRank to the citation analysis," *Inf. Process. Manage.*, vol. 44, no. 2, pp. 800–810, 2008.

[82] M. Franceschet, "PageRank: Standing on the shoulders of giants," *Commun. ACM*, vol. 54, no. 6, pp. 92–101, 2011.

[83] S. Ghosh, B. Viswanath, F. Kooti, N. K. Sharma, G. Korlam, F. Benevenuto, N. Ganguly, and K. P. Gummadi, "Understanding and combating link farming in the Twitter social network," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 61–70.

[84] Y. Chang, X. Wang, Q. Mei, and Y. Liu, "Towards Twitter context summarization with user influence models," in *Proc. 6th ACM Int. Conf. Web Search Data Mining*, 2013, pp. 527–536.

[85] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 199–208.

[86] K. Avrachenkov, et al., "Monte Carlo methods in PageRank computation: When one iteration is sufficient," *SIAM J. Numerical Anal.*, vol. 45, no. 2, pp. 890–904, 2007.

[87] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using PageRank vectors," in *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci.*, 2006, pp. 475–486.

[88] M. Kas, M. Wachs, K. M. Carley, and L. R. Carley, "Incremental algorithm for updating betweenness centrality in dynamically growing networks," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2013, pp. 33–40.

[89] G. Ramalingam and T. Reps, "On the computational complexity of dynamic graph problems," *Theoretical Comput. Sci.*, vol. 158, no. 1/2, pp. 233–277, 1996.

[90] P. Bisenius, E. Bergamin, E. Angriman, and H. Meyerhenke, "Computing top-k closeness centrality in fully-dynamic graphs," in *Proc. 20th Workshop Algorithm Eng. Experiments*, 2018, pp. 21–35.

[91] E. Nathan, G. Sanders, J. Fairbanks, D. A. Bader, et al., "Graph ranking guarantees for numerical approximations to katz centrality," *Procedia Comput. Sci.*, vol. 108, pp. 68–78, 2017.

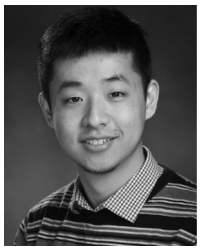
[92] A. van der Grinten, E. Bergamini, O. Green, D. A. Bader, and H. Meyerhenke, "Scalable katz ranking computation in large static and dynamic graphs," *26th Annu. Eur. Symp. Algorithms*, 2018, pp. 42:1–42:14.

[93] M. Riondato and E. Upfal, "ABRA: Approximating betweenness centrality in static and dynamic graphs with rademacher averages," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 5, 2018, Art. no. 61.

[94] F. Bonchi, G. De Francisci Morales, and M. Riondato, "Centrality measures on big graphs: Exact, approximated, and distributed algorithms," in *Proc. 25th Int. Conf. Companion World Wide Web*, 2016, pp. 1017–1020.

[95] M. Kimura and K. Saito, "Tractable models for information diffusion in social networks," in *Proc. Eur. Conf. Principles Data Mining Knowl. Discovery*, 2006, pp. 259–271.

[96] C. Zhou, et al., "UBLF: An upper bound based approach to discover influential nodes in social networks," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2013, pp. 907–916.
[97] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 695–710.
[98] S. Vaswani, L. Lakshmanan, M. Schmidt, et al., "Influence maximization with bandits," *arXiv:1503.00024*, 2015.
[99] P. Lagrée, O. Cappé, B. Cautis, and S. Maniu, "Algorithms for online influencer marketing," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 1, 2018, Art. no. 3.
[100] M. G. Rodriguez and B. Schölkopf, "Influence maximization in continuous time diffusion networks," in *Proc. 29th Int. Conf. Mach. Learning*, 2012, pp. 579–586.
[101] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Proc. Int. School Formal Methods Des. Comput. Commun. Softw. Syst.*, 2007, pp. 220–270.
[102] A. Goyal, et al., "A data-based approach to social influence maximization," *Proc. VLDB Endowment*, vol. 5, no. 1, pp. 73–84, 2011.
[103] M. G. Rodriguez, J. Leskovec, D. Balduzzi, and B. Schölkopf, "Uncovering the structure and temporal dynamics of information propagation," *Netw. Sci.*, vol. 2, no. 1, pp. 26–65, 2014.
[104] H. Daneshmand, M. Gomez-Rodriguez, L. Song, and B. Schölkopf, "Estimating diffusion network structures: Recovery conditions, sample complexity & soft-thresholding algorithm," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 793–801.
[105] M. Gomez Rodriguez, J. Leskovec, and A. Krause, "Inferring networks of diffusion and influence," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 1019–1028.
[106] M. Tahani, A. Hemmatyar, H. R. Rabiee, and M. Ramezani, "Inferring dynamic diffusion networks in online media," *ACM Trans. Knowl. Discovery Data*, vol. 10, no. 4, 2016, Art. no. 44.
[107] Z. Wang, Y. Yang, J. Pei, L. Chu, and E. Chen, "Activity maximization by effective information diffusion in social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2374–2387, Nov. 2017.
[108] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "DeepInf: Social influence prediction with deep learning," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 2110–2119.
[109] M. Farajtabar, Y. Wang, M. G. Rodriguez, S. Li, H. Zha, and L. Song, "COEVOLVE: A joint point process model for information diffusion and network co-evolution," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1954–1962.



Yu Yang received the BE degree from the Hefei University of Technology, in 2010, the ME degree from the University of Science and Technology of China, in 2013, both in computer science, and the PhD degree in computing science from Simon Fraser University, in Feb. 2019. He is currently an assistant professor with the School of Data Science, City University of Hong Kong. His research interests lie in the algorithmic aspects of data mining and data science, with an emphasis on managing and mining dynamics of large scale networks.

Jian Pei (Fellow) is a professor with the School of Computing Science, Simon Fraser University, Canada. His research interests can be summarized as developing effective and efficient data analysis techniques for novel data intensive applications. He is currently interested in various techniques of data mining, Web search, information retrieval, data warehousing, online analytical processing, and database systems, as well as their applications in social networks, health-informatics, business and bioinformatics. His research has been supported in part by government funding agencies and industry partners. He has published prolifically and served regularly for the leading academic journals and conferences in his fields. He is a fellow of the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE). He is the recipient of several prestigious awards.