On the Eigen-Functions of Dynamic Graphs: Fast Tracking and Attribution Algorithms

Chen Chen* and Hanghang Tong

Department of Computer Science, Arizona State University, Tempe, AZ 85281, USA

Received 10 September 2015; revised 6 January 2016; accepted 5 February 2016
DOI:10.1002/sam.11310
Published online in Wiley Online Library (wileyonlinelibrary.com).

Abstract: Eigen-functions are of key importance in graph mining since they can be used to approximate many graph parameters, such as node centrality, epidemic threshold, graph robustness, with high accuracy. As real-world graphs are changing over time, those parameters may get sharp changes correspondingly. Taking virus propagation network for example, new connections between infected and susceptible people appear all the time, and some of the crucial infections may lead to large decreasing on the epidemic threshold of the network. As a consequence, the virus would spread around the network quickly. However, if we can keep track of the epidemic threshold as the graph structure changes, those crucial infections would be identified timely so that counter measures can be taken proactively to contain the spread process. In our paper, we propose two online eigen-functions tracking algorithms which can effectively monitor those key parameters with linear complexity. Furthermore, we propose a general attribution analysis framework which can be used to identify important structural changes in the evolving process. In addition, we introduce an error estimation method for the proposed eigen-functions tracking algorithms to estimate the tracking error at each time stamp. Finally, extensive evaluations are conducted to validate the effectiveness and efficiency of the proposed algorithms. © 2016 Wiley Periodicals, Inc. Statistical Analysis and Data Mining: The ASA Data Science Journal, 2016

Keywords: dynamic graph; connectivity; graph spectrum

1. INTRODUCTION

To better understand the node centrality and connectivity of graphs, various graph parameters have been invented for different tasks. Though different on their definitions, many of those parameters can be well approximated by some well defined eigen-functions. For example, in node centrality analysis, one commonly used parameter is eigenvector centrality [1], which is defined with the leading eigenvector of the graph. As for graph connectivity, frequently used parameters include epidemic threshold ([2–4]), clustering coefficient [5], graph robustness ([6–8]), eigen-gap, etc. For epidemic threshold, Prakash et al. found that the tipping point for the dissemination process in *arbitrary* graph is controlled by the leading eigenvalue of certain system matrix associated with the graph [3]. For clustering coefficient calculation, the most time consuming part is counting the number of triangles in the graph, which is of $O(n^3)$ complexity. In ref. [9], Tsourakakis proved that the number of triangles in a graph can be accurately estimated with its top eigenvalues. Similar to clustering coefficient, Chan et al. showed that natural connectivity [10], a good measurement for graph robustness, can also be approximated with the top eigenvalues of the graph. Moreover, as shown in ref. [11], the expansion property of a graph can be measured with its eigen-gap between first and second eigenvalues [12].

Most of the graph parameters mentioned above are all based on static graphs. However in real-world applications, the graph structure evolves over time. In some cases, subtle changes on the graph structure may lead to huge difference on some of its properties. For example, when Ebola virus was first brought to the US continent, some emerging connections in the contact network would greatly reduce the epidemic threshold of the graph, and eventually cause the outbreak of the disease. By monitoring those key parameters as graph evolves and analyzing the attribution for sharp parameter changes timely, we would be able to get prepared for emergent events at an early stage. Another application scenario is social network. In websites

* *Correspondence to:* Chen Chen (chen_chen@asu.edu)

like Facebook and Twitter, new connections between users emerge all the time, which would in turn change the influential individuals in the network. It is crucial for online marketing companies to keep track of those changes since their advertisements targeting strategies may need to be modified accordingly.

For eigen-functions tracking problem, simply re-computing the eigen-pairs whenever the graph structure changes is computationally costly over fast-changing large graphs. The popular Lanczos method for computing top-k eigen-pairs would require $O(mk + nk^2)$ time, where m and n are the numbers of edges and nodes in the graph, respectively. Although the complexity seems acceptable for one-time calculation in static graphs, it would be too expensive for large dynamic graphs. To address this challenge, we consider a way of updating the eigen-pairs incrementally instead of re-computing them from scratch at each time stamp. In this paper, we propose two online algorithms to track the eigen-pairs of a dynamic graph efficiently, which bear linear time complexities with respect to the number of nodes n in the graph and the number of changed edges s at current stamp. Based on these algorithms, we introduce a general attribution analysis framework for identifying key connection changes that have largest impact on the graph. Last, to control the accumulated tracking error of eigen-functions, we propose an error estimation method to detect sharp error increase timely so that the accumulated error can be eliminated by restarting the tracking algorithms.

In addition to the problem definition, the main contributions of this paper can be summarized as follows:

- *Algorithms*. We propose two online algorithms to track the top eigen-pairs of a dynamic graph, which in turn enable us to track a variety of important network parameters based on certain eigen-functions. In addition, we provide a framework for attribution analyses on eigen-functions and a method for estimating tracking errors.

- *Evaluations*. We evaluate our methods with other eigen-pair update algorithms on real-world datasets, to validate the effectiveness and efficiency of the proposed algorithms.

The rest of the paper is organized as follows: In Section 2, a brief survey of related studies on graph spectrum, dynamic graphs and general graph mining is provided. A formal problem definition is given in Section 3. Section 4 gives the first-order and high-order eigen-functions tracking algorithms, attribution analysis framework, error estimation algorithm, and corresponding analysis. Experimental results are shown in Section 5 and we conclude in Section 6.

2. RELATED WORK

Eigen-pairs of a graph can be derived to various important parameters, which can be used to describe the graph from different aspects. Those parameters are widely used for different graph mining tasks. Tracking those eigen-functions on fast-changing dynamic graphs can be abstracted as a process of conducting evolutionary analysis on streaming networks. Here we organize our related work into three sections: (A) work on applications of different eigen-functions; (B) work on dynamic graph analysis; and (C) general graph mining.

2.1. Applications of Eigen-Functions

According to ref. [11], the eigen-pairs on adjacency matrix and those on Laplacian matrix of a graph have different meanings. The eigenvalues of adjacency matrix can be used to determine the path capacity of a graph [13], while for Laplacian matrix, they indicate the connectivity of the graph. Based on these two meanings, a large amount of work was developed regarding to path capacity and graph connectivity, respectively.

In refs. [9,14], Tsourakakis found that the total number of triangles in the graph and number of triangles that contain certain node can be efficiently estimated with the eigenvalues of graph adjacency matrix. In refs. [2,15], Ganesh et al. and Chakrabati et al. proved that the epidemic threshold for SIS model on arbitrary undirected network is related to the leading eigenvalue of graph adjacency matrix. Prakash et al. further improved their work by proving that the threshold for a variety of cascade models on arbitrary network depend on the first eigenvalue of certain system matrix associated with the network[3]. Tong et al. proposed a node manipulation method in ref. [16] and edge manipulation method in ref. [17] to optimize the change of first eigenvalue in the graph. In Le et al. [18] found that most eigenvalue optimization methods perform poorly when the eigen-gap [11] of the graph is small and proposed MET algorithm for eigenvalue minimization in such "small gap" graphs. In, Hoory et al. [12] introduced the concept of expansion property of the graph, which inspired many network robustness related works [19,20]. In, Chan et al. [7] proposed a more general robustness measurement and provided corresponding graph manipulating strategies (on both nodes and edges) to optimize the robustness score. Recently, they showed that eigen-pairs can also be used to locate robust subgraphs in the network in [21]. On the other hand for Laplacian matrix of the graph, Newman showed that the eigen-pairs of Laplacian matrix can be used for community detection [22,23]. In our work, we will focus on the eigen-functions of graph adjacency matrix.

2.2. Dynamic Graphs Analysis

Dynamic graph analysis has attracted much attention in recent years. Aggarwal and Subbian have made a thorough summary of related research in ref. [24]. The research on dynamic graph analysis can be generally sorted into two categories: (A) monitoring the change on the evolving graph and (B) efficiently updating the data mining results as graph changes.

In Leskovec et al. [25,26] discovered the growth pattern of real graphs by their densities and diameters. As graph mining tasks vary from one another, the parameters tracked in the process are different. In ref. [27], two online algorithms were provided for tracking node proximity and centrality on bipartite graphs. In, Malliaros et al. [28] defined a new graph robustness property based on top k eigen-pairs of the graph, and proposed an algorithm to detect communities and anomalies. Similar mechanism for anomaly detection was used in ref. [29] based on eigen-pairs of dependency matrix of the graph. Ferlez et al. [30] proposed a dynamic graph monitoring algorithm based on MDL (Minimum Description Length) [31] which can be used to detect the changing communities in the evolving process. In ref. [32], a graph kernel tracking algorithm was proposed for dynamic graphs. The other area of research that is remotely related to our work is evolutionary spectral clustering on graphs. In, Ning et al. [33] proposed an incremental spectral clustering algorithm based on iterative update on the eigen-system of the graph.

2.3. General Graph Mining

Graph mining has been a hot research topic for years. Depending on the type of the graph, graph mining related research can be classified into two categories: (1) single-layered graph mining and (2) multi-layered graph mining. For single-layered graphs, classic works include pattern and law mining [34–36], frequent substructure discovery [37–39], community mining and graph partition [40,41], proximity [42–44], graph sampling [45,46], information propagation [47–49], etc. As for multi-layered graphs, one of the key problem being studied is cascading failure in interdependent system [50,51]. In refs. [52–55], different types of *two-layered* interdependent networks were thoroughly analyzed. In refs. [56,57], different kinds of more generally structured multi-layered networks were studied.

3. PROBLEM DEFINITION

In this section, we introduce the notations used through out the paper, and four important eigen-functions in graph mining, followed by a formal definition of eigen-functions tracking problem.

Table 1. Symbols used in text.

Symbol	Definition and Description
$G^t(V, E)$	undirected, unipartite network at time t
m	number of edges in the network
n	number of nodes in the network
$\mathbf{B}, \mathbf{C}, \ldots$	matrices (bold upper case)
$\mathbf{b}, \mathbf{c}, \ldots$	vectors (bold lower case)
$\mathbf{A}^t$	adjacency matrix of $G^t(V, E)$ at time t
$\Delta\mathbf{A^t}$	perturbation matrix from time t to $t+1$
$\triangle(G^t)$	number of triangles in G^t
$S(G^t)$	robustness score of G^t
$Gap(G^t)$	eigen-gap of G^t
$(\lambda_j{}^t, \mathbf{u_j}^t)$	j^{th} eigen-pair of $\mathbf{A}^t$
$[\Delta\mathbf{A}^t]_{t=t_1\ldots t_2}$	perturbation matrices of dynamic graph from time t_1 to t_2
$[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1\ldots t_2}$	top k eigen-pairs from time t_1 to t_2
$[\triangle(G^t)]_{t=t_1\ldots t_2}$	$\triangle(G)$ from time t_1 to t_2
$[S(G^t)]_{t=t_1\ldots t_2}$	$S(G)$ from time t_1 to t_2

3.1. Notations

The symbols used throughout the text is shown in Table 1. We consider the graph in each time stamp $G^t(V, E)$ is undirected and unipartite. In consistent with standard notation, we use bold upper-case for matrices (e.g., $\mathbf{B}$), and bold lower-case for vectors (e.g., $\mathbf{b}$). For each time stamp, the graph is represented by its adjacency matrix $\mathbf{A}^t$. $\Delta\mathbf{A}^t$ denotes the perturbation matrix from time t to $t+1$. $(\lambda_j^t, \mathbf{u_j}^t)$ is the j^{th} eigen-pair of $\mathbf{A}^t$. The number of triangles and robustness score of the graph at time t are represented as $\triangle(G^t)$ and $S(G^t)$, respectively.

With the above notations, the eigen-function is defined as a function that maps eigen-pairs of the graph to certain graph attribute or attribute vector, which can be expressed as

$$\mathrm{f} : (\Lambda_k, \mathbf{U}_k) \rightarrow \mathbb{R}^x (x \in \mathbb{N}) \tag{1}$$

3.2. Important Eigen-Functions

3.2.1. Eigenvalues and Eigenvectors

Since the eigen-pairs of a graph are important attibutes themselves, the simplest eigen-function is therefore an identity function as follows:

$$f((\Lambda_k, \mathbf{U}_k)) = (\Lambda_k, \mathbf{U}_k) \tag{2}$$

The eigenvalues of a graph's adjacency matrix can be used to measure the path capacity of the graph [13], while the eigenvectors can be used to evaluate the centrality of nodes [1], or to detect interesting subgraphs [58].

In most of the applications, only top k (k varies under different settings) eigen-pairs $(\Lambda_k{}^t, \mathbf{U}_k^t)$ are used. Therefore it is not necessary to compute the complete set of eigen-pairs in real analysis.

3.2.2. Number of Triangles in Graph

The number of triangles in a graph plays an important role in calculating clustering coefficient and related attributes. The brute-force algorithm for solving this problem is of complexity $O(n^3)$. State-of-the-art algorithm has reduced the complexity to $O(n^{2.373})$ [59], but this is still not a scalable algorithm on real-world large datasets. In, Tsourakakis proposed [9] a fast triangle counting algorithm which showed that the number of triangles in a graph($\triangle(G)$) can be estimated using Eq. (3).

$$f((\Lambda_k, \mathbf{U}_k)) = \triangle(G) = \frac{1}{6}\sum_{i=1}^{k}\lambda_i^3 \tag{3}$$

By Eq. (3), number of triangles $\triangle(G)$ therefore becomes a function of eigenvalues Λ_k. Again, for real-world graphs, usually, we only need top k eigenvalues to achieve a good approximation for triangle counting. For example, experiments in ref. [9] showed that picking top 30 eigen-pairs can achieve an accuracy of at least 95% in most graphs.

3.2.3. Robustness Measurement

The robustness score of a network evaluates it tolerance under error and external attacks. Although there are many kinds of robustness measurements being used in graph analysis, few of them can act as an universal standard that can fully express the resilience of the network from different points of view. Chan et al. provided a thorough analysis of different robustness measurements and proposed the idea of using *natural connectivity* as robustness score, which overcomes most of the shortcomings that previous measurements have [7]. The definition of robustness score($S(G)$) [7] is shown in Eq. (4).

$$f((\Lambda_k, \mathbf{U}_k)) = S(G) = \ln(\frac{1}{k}\sum_{j=1}^{k} e^{\lambda_j}) \tag{4}$$

By Eq. (4), robustness score $S(G)$ is also a function of eigenvalues Λ_k.

Once again, In, Chan et al. [7] found that top k ($k = 50$ in their study) eigen-pairs are sufficient for estimating robustness score.

3.2.4. Eigen-Gap

The eigen-gap of a graph is an important parameter in expander graph theory and is defined as the difference between the largest and second largest (in module) eigenvalues of the graph (as shown in Eq. (5)).

$$f((\Lambda_k, \mathbf{U}_k)) = Gap(G) = \lambda_1 - \lambda_2 \tag{5}$$

In expander graph theory, a graph is considered to have a good expansion property if it is both sparse and highly connected [12]. By Cheeger inequality, the expansion property of a graph is strongly correlated to its eigen-gap [11]. As a result, the eigen-gap of the graph can be used as another measurement for its robustness.

3.3. Problem Definition

In all the above cases, the network parameters of interest (e.g., epidemic threshold, eigen centrality, number of triangles, robustness measurement, eigen-gap) can always be expressed as functions of eigen-pairs of the underlying graph. What is more, for real graphs, it is often sufficient to use top-k eigen-pairs to achieve a high accuracy estimation of these parameters. Therefore, in order to track these parameters on a dynamic graph, we only need to track the corresponding top-k eigen-pairs at each time stamp. Formally, the eigen-function tracking problem is defined as follows. Once the top-k eigen-pairs are estimated, we can use Equ (2) to (5) to update the corresponding eigen-functions.

Problem 1 ***Top*-k *Eigen-Pairs Tracking***

Given: *(1) a dynamic graph G tracked from time t_1 to t_2 with starting matrix $\mathbf{A}^{t_1}$, (2) an integer k, and (3) a series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$;*

Output: *the corresponding top-*k *eigen-pairs at each time stamp $[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1,\ldots,t_2}$.*

4. TRIP: TRACKING EIGEN-PAIRS

In this section, we present our solutions for Problem 1. We start with a baseline solution (TRIP-BASIC), and then present its high-order variant (TRIP), followed by the attribution analysis framework for different eigen-functions and an error estimation method.

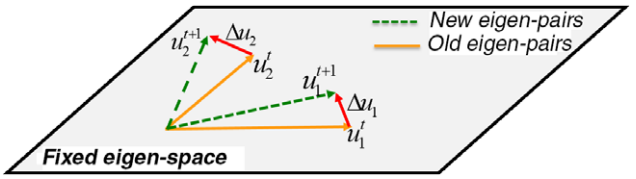

Fig. 1 Incremental update for eigen-pairs tracking. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

4.1. Key Idea

The key idea for TRIP-BASIC and TRIP is to incrementally update the eigen-pairs with corresponding perturbation terms at each time stamp. By matrix perturbation theory[60], we have the following perturbation equation

$$(\mathbf{A}^t + \Delta\mathbf{A}^t)(\mathbf{u_j}^t + \Delta\mathbf{u_j}) = (\lambda_j^t + \Delta\lambda_j)(\mathbf{u_j}^t + \Delta\mathbf{u_j}) \quad (6)$$

As the perturbation matrix is often very sparse, it is natural to assume that graphs in two consecutive time stamps share a fixed eigen-space. Therefore, the perturbation eigenvector $\Delta\mathbf{u_j}$ can be expressed as $\Delta\mathbf{u_j} = \sum_{i=1}^{k} \alpha_{ij}\mathbf{u_i}^t$, which is the linear combination of old eigenvectors. Taking the two dimensional eigen-space in Fig. 1 as an example, the old eigenvectors are $\mathbf{u_1}^t$ and $\mathbf{u_2}^t$ marked in orange; the new eigenvectors $\mathbf{u_1}^{t+1}$ and $\mathbf{u_2}^{t+1}$ (in green) can be decomposed into old eigenvectors $\mathbf{u_1}^t$, $\mathbf{u_2}^t$ and perturbation eigenvectors $\Delta\mathbf{u_1}$, $\Delta\mathbf{u_2}$ in the same plane.
Expanding Eq. (6), we get

$$\begin{aligned}&\mathbf{A}^t\mathbf{u_j}^t + \Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{A}^t\Delta\mathbf{u_j} + \Delta\mathbf{A}^t\Delta\mathbf{u_j}\\ &\quad = \lambda_j^t\mathbf{u_j}^t + \Delta\lambda_j\mathbf{u_j}^t + \lambda_j^t\Delta\mathbf{u_j} + \Delta\lambda_j\Delta\mathbf{u_j}\end{aligned}$$

By the fact that $\mathbf{A}^t\mathbf{u_j}^t = \lambda_j^t\mathbf{u_j}^t$, the perturbation equation can be simplified as

$$\Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{A}^t\Delta\mathbf{u_j} + \Delta\mathbf{A}^t\Delta\mathbf{u_j} = \Delta\lambda_j\mathbf{u_j}^t + \lambda_j^t\Delta\mathbf{u_j} + \Delta\lambda_j\Delta\mathbf{u_j} \quad (7)$$

Multiplying the term $\mathbf{u_j}^{t'}$ on both sides; as eigenvectors are of unit length, we have

$$\mathbf{u_j}^{t'}\Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{u_j}^{t'}\Delta\mathbf{A}^t\Delta\mathbf{u_j} = \Delta\lambda_j + \mathbf{u_j}^{t'}\Delta\lambda_j\Delta\mathbf{u_j} \quad (8)$$

As we assume that $\Delta\mathbf{u_j} \ll \mathbf{u_j}$ and $\Delta\lambda_j \ll \lambda_j$, the high order terms $\mathbf{u_j}^{t'}\Delta\mathbf{A}^t\Delta\mathbf{u_j}$ and $\mathbf{u_j}^{t'}\Delta\lambda_j\Delta\mathbf{u_j}$ in Eq. (8) can be discarded without losing too much accuracy. Therefore, $\Delta\lambda_j$ can be estimated as

$$\Delta\lambda_j = \mathbf{u_j}^{t'}\Delta\mathbf{A}^t\mathbf{u_j}^t \quad (9)$$

The difference between TRIP-BASIC and TRIP lies in their ways of estimating perturbation eigenvectors, which will be discussed in the following subsections.

4.2. TRIP-BASIC

TRIP-BASIC is a first-order eigen-pair tracking method, which ignores the high-order terms in the perturbation equation when updating eigenvectors at each time stamp. By removing the high-order terms, the perturbation equation Eq. (7) can be written as

$$\Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{A}^t\Delta\mathbf{u_j} = \Delta\lambda_j\mathbf{u_j}^t + \lambda_j^t\Delta\mathbf{u_j}$$

Replacing all $\Delta\mathbf{u_j}$ terms with $\sum_{i=1}^{k}\alpha_{ij}\mathbf{u_i}^t$ and multiplying the term $\mathbf{u_p}^{t'}$ ($p \neq j$) on both sides, by applying the orthogonality property of eigenvectors to the new equation, we can solve the coefficient α_{pj} as

$$\alpha_{pj} = \frac{\mathbf{u_p}^{t'}\Delta\mathbf{A}^t\mathbf{u_j}^t}{\lambda_j^t - \lambda_p^t}$$

Therefore $\Delta\mathbf{u_j}$ can estimated as

$$\Delta\mathbf{u_j} = \sum_{i=1, i\neq j}^{k} \left(\frac{\mathbf{u_i}^{t'}\Delta\mathbf{A}^t\mathbf{u_j}^t}{\lambda_j^t - \lambda_i^t}\mathbf{u_i}^t\right) \quad (10)$$

Suppose $\mathbf{A}^t$ is perturbed with a set of edges $\Delta E = < p_1, r_1 >, \ldots, < p_s, r_s >$ where s is the number of non-zero elements in perturbation matrix $\Delta\mathbf{A}$. In Eq. (10), the term $\mathbf{u_j}^{t'}\Delta\mathbf{A}\mathbf{u_j}^t$ can be expanded as

$$\mathbf{u_j}^{t'}\Delta\mathbf{A}^t\mathbf{u_j}^t = \sum_{<p,r>\in\Delta E} \Delta\mathbf{A}^t(p,r)\mathbf{u_{pj}}^t\mathbf{u_{rj}}^t \quad (11)$$

Eqs. (10) and (11) naturally lead to our base solution (TRIP-BASIC) for solving Problem 1 as follows.

The approximated eigen-pairs for each time stamp is computed from steps 2 to 10. Each $\Delta\lambda_j$ and $\Delta\mathbf{u}_j$ is calculated from steps 3 to 7 by Eqs. (10) and (11). At steps 8 and 9, λ_j^t and $\mathbf{u_j}^t$ is updated with $\Delta\lambda_j$ and $\Delta\mathbf{u}_j$. Note that after updating the eigenvector in step 9, we normalize each of them to unit length.

4.2.1. *Complexity Analysis*

The efficiency of proposed Algorithm 1 is summarized in Lemma 1. Both time complexity and space complexity is linear with respect to the total number of the nodes in the graph (n) and total number of the time stamps (T).

LEMMA 1: ***Complexity of First Order Eigen-Function Tracking.*** Suppose T is the total number of the time

Algorithm 1 TRIP-BASIC: First Order Eigen-Pairs Tracking

Input: Dynamic graph G tracked from time t_1 to t_2, with starting eigen-pairs $({\Lambda_k}^{t_1}, \mathbf{U}_k^{t_1})$, series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$

Output: Corresponding eigen-pairs $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1,\ldots t_2}$

1: **for** $t = t_1$ to $t_2 - 1$ **do**
2: **for** $j = 1$ to k **do**
3: Initialize $\Delta\mathbf{u_j} \leftarrow \mathbf{0}$
4: **for** $i = 1$ to k,$i \neq j$ **do**
5: $\Delta\mathbf{u_j} \leftarrow \Delta\mathbf{u_j} + \frac{\mathbf{u_i^{t\prime}}\Delta\mathbf{A^t}\mathbf{u_j^t}}{\lambda_j^t - \lambda_i^t}\mathbf{u_i^t}$
6: **end for**
7: Calculate $\Delta\lambda_j \leftarrow \mathbf{u_j^{t\prime}}\Delta\mathbf{A^t}\mathbf{u_j^t}$
8: Update $\lambda_j^{t+1} \leftarrow \lambda_j^t + \Delta\lambda_j$
9: Update ${\mathbf{u_j}}^{t+1} \leftarrow {\mathbf{u_j}}^t + \Delta\mathbf{u_j}$
10: **end for**
11: **end for**
12: Return $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1\ldots t_2}$

stamps, s is the average number of perturbed edges in $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$, then the time cost for Algorithm 1 is $O(Tk^2(s+n))$; the space cost is $O(Tnk+s)$.

Proof: In each time stamp from time t_1 to $t_2 - 1$, top k eigen-pairs are updated in steps 2-10. By Eq. (11), the complexity of computing term $\mathbf{u_j}^{t\prime}\Delta\mathbf{A}^t\mathbf{u_j}^t$ is $O(s)$, so the overall complexity of step 5 is $O(s+n)$. Therefore calculating $\Delta\mathbf{u_j}$ from steps 4 to 6 takes $O(k(s+n))$. In step 7, computing $\Delta\lambda_j$ takes another $O(s)$. Updating λ_j^t and ${\mathbf{u_j}}^t$ in step 8 and 9 takes $O(1)$ and $O(n)$. Therefore updating all top-k eigen-pairs ${\mathbf{U_k}}^t$ and Λ_k^t takes $O(k^2(s+n))$ and $O(ks)$, respectively. Thus the overall time complexity for T iterations is $O(Tk^2(s+n))$.
For space cost, it takes $O(k)$ and $O(nk)$ to store Λ_k^t and ${\mathbf{U_k}}^t$ at each time stamp. In the update phase from step 2 to 10, it takes $O(s)$ to store $\Delta\mathbf{A}^t$, $O(1)$ to update λ_j^t and $O(n)$ to update ${\mathbf{u_j}}^t$. However the space used in the update phase can be reused in each iteration. Therefore the overall space complexity for T time stamps takes a space of $O(Tnk+s)$. ■

4.3. TRIP

The baseline solution in Algorithm 1 is simple and straight-forward, but it has the following limitations. First, the approximation error of first order matrix perturbation is in the order of $\|\Delta\mathbf{A}^t\|$. In other words, the quality of such approximation might decrease quickly with respect to the increase of $\|\Delta\mathbf{A}^t\|$. Second, the approximation quality is highly sensitive to the small eigen-gap of $\mathbf{A}^t$ as indicated by Eq. (10). In order to address these limitations, we further propose Algorithm 2 by adopting the high-order matrix perturbation to update the eigen-pairs of $\mathbf{A}^{t+1}$. The main difference between Algorithm 2 and Algorithm 1 is that we take high-order terms in the perturbation equation (Eq. (7)) into consideration while updating eigenvectors. Similar to TRIP-BASIC we replace all $\Delta\mathbf{u_j}$ terms with $\sum_{i=1}^k \alpha_{ij}{\mathbf{u_i}}^t$ and multiplying the term ${\mathbf{u_p}}^{t\prime}$ (for $1 \leq p \leq k,\ p \neq j$) on both sides. By applying the orthogonality property of eigenvectors to the new equation, we have

$$\mathbf{X}^t(p,j) + \alpha_{pj}\lambda_p^t + \sum_{i=1}^{k}\mathbf{X}^t(p,i)\alpha_{ij} = \alpha_{pj}\lambda_j^t + \alpha_{pj}\Delta\lambda_j$$

where $\mathbf{X}^t = {\mathbf{U}_k^t}'\Delta\mathbf{A}^t\mathbf{U}_k^t$. Reorganizing the terms in the above equation, we have

$$\mathbf{X}^t(p,j) - \alpha_{pj}(\lambda_j^t + \Delta\lambda_j - \lambda_p^t) + \sum_{i=1}^{k}\mathbf{X}^t(p,i)\alpha_{ij} = 0$$

By defining $\mathbf{v} = \lambda_j^t + \Delta\lambda_j - \lambda_p^t$ for $p = 1,\ldots,k$, $\mathbf{D}^t = diag(\mathbf{v})$ and $\alpha_\mathbf{j} = [\alpha_{1j},\ldots,\alpha_{kj}]$, the above equation can be expressed as

$$\mathbf{X}^t(:,j) - \mathbf{D}^t\alpha_\mathbf{j} + \mathbf{X}^t\alpha_\mathbf{j} = \mathbf{0}$$

Solve the above equation for $\alpha_\mathbf{j}$, we have

$$\alpha_\mathbf{j} = (\mathbf{D}^t - \mathbf{X}^t)^{-1}\mathbf{X}^t(:,j)$$

Algorithm 2 TRIP: High Order Eigen-Pairs Tracking

Input: Dynamic graph G tracked from time t_1 to t_2, with starting eigen-pairs $({\Lambda_k}^{t_1}, \mathbf{U}_k^{t_1})$, series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$

Output: Corresponding eigen-pairs $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1,\ldots t_2}$

1: **for** $t = t_1$ to $t_2 - 1$ **do**
2: Calculate $\mathbf{X}^t \leftarrow {\mathbf{U}_k^t}'\Delta\mathbf{A}^t\mathbf{U}_k^t$
3: $\Delta\Lambda_k \leftarrow diag(X^t)$[1]
4: Update $\Lambda_k^{t+1} \leftarrow \Lambda_k^t + \Delta\Lambda_k$
5: **for** $j = 1$ to k **do**
6: Calculate $\mathbf{v} \leftarrow \lambda_j^t + \Delta\lambda_j - \lambda_p^t$ for $p = 1,\ldots,k$
7: $\mathbf{D}^t \leftarrow diag(\mathbf{v})$
8: Calculate $\alpha_\mathbf{j} \leftarrow (\mathbf{D}^t - \mathbf{X}^t)^{-1}\mathbf{X}(:,j)$
9: Calculate $\Delta\mathbf{u_j} \leftarrow \sum_{i=1}^k \alpha_{ij}{\mathbf{u_i}}^t$
10: Update ${\mathbf{u_j}}^{t+1} \leftarrow {\mathbf{u_j}}^t + \Delta\mathbf{u_j}$
11: **end for**
12: **end for**
13: Return $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1\ldots t_2}$

[1] Here the $diag$ function works the same with the one in Matlab. When apply to a matrix, $diag$ returns a vector of the main diagonal elements of the matrix; when apply to a vector, it returns a square diagonal matrix with the elements of vector on the main diagonal.

In Algorithm 2, the top-k eigen-pairs at each time stamp is updated from step 2 to 11. In step 2, matrix $\mathbf{X}^t$ is calculated for computing $\Delta\Lambda_k$ and $\Delta\mathbf{U}_k$. In step 4, all top-k eigenvalues Λ_k are updated by $\Delta\Lambda_k$. From step 6 to 10, each $\mathbf{u_j}^t$ is updated according to the derivations of the eigen update rule in mentioned above. Again, after we update the eigenvectors in step 9, we normalize each of them to unit length.

Complexity Analysis The efficiency of Algorithm 2 is given in Lemma 2. Compared with TRIP-BASIC, both time and space complexity are still linear with respect to total number of nodes in the graph and total number of time stamps, with a slight increase in k, which is often very small.

LEMMA 2: ***Complexity of High Order Eigen-Function Tracking.*** Suppose T is the total number of time stamps, s is the average number of perturbed edges in $[\Delta\mathbf{A^t}]_{t=t_1,...t_2-1}$, then the time cost for Algorithm 2 is $O(T(k^4+k^2(n+s)))$; the space cost is $O(Tnk+k^2+s)$.

Proof: In each time stamp from time t_1 to t_2-1, top k eigen-pairs are updated in steps 2-11. Using the update rule provided in Eq. (11), calculating $\mathbf{X}^t$ in step 2 takes $O(k^2s)$. Updating top eigenvalues in step 3-4 takes $O(k)$. From step 5 to 11, eigenvectors are updated. It takes $O(k^3)$ in to do matrix inversion and multiplication in step 8 and $O(nk)$ to calculate $\Delta\mathbf{u_j}$ in step 9. Therefore updating $\mathbf{U}_k^t$ takes $O(k^4+nk^2)$). Thus the overall time complexity for T iterations takes $O(T(k^4+k^2(n+s)))$.
For space cost, it takes $O(k)$ and $O(nk)$ to store Λ_k^t and $\mathbf{U_k}^t$, $O(s)$ to store $\Delta\mathbf{A}^t$ for each time stamp. In the update phase from step 2 to 11, it takes $O(k^2)$ to store and calculate $\mathbf{X}^t$, $\mathbf{D}^t$; $O(k)$ to store v and $\alpha_\mathbf{j}$; $O(k^2)$ to calculate $\alpha_\mathbf{j}$. However the space cost in update phase can be reused in each iteration. Therefore the overall space complexity for T time stamps takes a space of $O(Tnk+k^2+s)$. ■

4.4. Attribution Analysis

Based on our TRIP algorithms, we can effectively track the corresponding eigen-functions of interest (as defined in subsection 3.2). In reality, we might also be interested in understanding the key factors that cause these changes in dynamic graphs. For example, among all the changed edges in $\Delta\mathbf{A}$, which edge is most important in causing the inrease/decrease of the epidemic threshold, or the number of triangles, etc. The importance of an edge $<p,r>\in\Delta E$ can be measured as the change it can make on the corresponding eigen-functions, which can be written as

$$score(<p,r>) \sim \Delta \mathrm{f}_{<p,r>} = f_{G\cup<p,r>} - f_G$$

where f(.) is one of eigen-functions we define in subsection 3.2.

Algorithm 3 Dynamic Attribution Analysis

Input: Dynamic graph G and eigen-pairs $({\Lambda_k}^t, \mathbf{U}_k^t)$ at time t, perturbation matrix $\Delta\mathbf{A}^t$, eigen-function $f(.)$, number l

Output: top l added edges and removed edges at time t that have largest impact on eigen-function $f(.)$

1: $removed \leftarrow$ extract all removed edges in $\Delta\mathbf{A}^t$
2: $added \leftarrow$ extract all added edges in $\Delta\mathbf{A}^t$
3: **for** each edge $<p,r>$ in $removed$ **do**
4: $\quad score(<p,r>) \leftarrow f_{G^t} - f_{G^t\setminus<p,r>}$
5: **end for**
6: **for** each edge $<p,r>$ in $added$ **do**
7: $\quad score(<p,r>) \leftarrow f_{G^t\cup<p,r>} - f_{G^t}$
8: **end for**
9: Return top l edges in $removed$ and $added$ with highest scores respectively

In Algorithm 3, all removed edges and added edges are extracted from $\Delta\mathbf{A}$ in steps 1 and 2. The impact score of each removed edge at time t is calculated from step 3 to 5. Similarly, the score of each added edge is calculated from step 6 to 8. At the end, top l removed edges and l added edges are returned as high impact edges at time t.

4.4.1. Complexity Analysis

Assume that the complexity of calculating $\Delta f_{<p,r>}$ is $h(n,k,s)$, where h is a function of number of nodes n, number of eigen-pairs k and number of changed edges s. Then the complexity of calculating the impact scores of all changed edges (from step 3 to 8) is $O(sh(n,k,s))$. Given the impact score of each changed edges, the complexity of picking out top l edges from $removed$ and $added$ set using heap structure is $O(|removed|\log l)+O(|added|\log l)=O(s\log l)$. Therefore the overall complexity for attribution analysis at time t is $O(s(h(n,k,s)+\log l))$.

4.5. Error Estimation

As described in section 4.1, the core mechanism for both TRIP-BASIC and TRIP is to incrementally update the eigen-pairs at each time stamp. With this scheme, the tracking error of eigen-pairs would accumulate as time goes by. Therefore, finding a proper time to restart the algorithm is of key importance to keep the tracking error within a reasonable range. For simplicity, we only estimate the error of leading eigenvalue since it is the key part for most of the eigen-functions. Here we denote $err(\lambda^t)$ as the estimated error on λ introduced at time t. Intuitively, $err(\lambda^t)$ would be strongly correlated to the impact of $\Delta\mathbf{A}^t$ on the original eigen-space. As the original eigen-space is defined by the

top-k eigenvectors $\mathbf{U}_k^{t_1}$ at the first time stamp t_1, to measure the impact of $\Delta\mathbf{A}^t$ on $\mathbf{U}_k^{t_1}$, we can project $\Delta\mathbf{A}^t$ into this space and take the Frobenius norm of the projection as its actual impact. Eq. (12) formalizes the impact function of $\Delta\mathbf{A}^t$ on eigen-space $\mathbf{U}_k^{t_1}$.

$$err(\lambda^t) \sim impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1}) = \| \mathbf{U}_k^{t_1}\mathbf{U}_k^{t_1\prime}\Delta\mathbf{A}^t \|_{Fro} \quad (12)$$

We denote the summation of the perturbation impacts from first time stamp t_1 to current stamp t as $err_{acc}(\lambda^t)$. This number can be viewed as a good approximation of accumulated tracking error on leading eigenvalue from t_1 to t. In other words, the curve of $err_{acc}(\lambda^t)$ from $t = t_1, \ldots, t_2$ would have similar shape with real tracking error curve of TRIP algorithms.

Algorithm 4 Error Estimation for Eigen-function Tracking

Input: Dynamic graph G tracked from time t_1 to t_2, with starting eigen-pairs $(\Lambda_k{}^{t_1}, \mathbf{U}_k^{t_1})$, series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$

Output: Corresponding estimated error $err_{acc}(\lambda^t)$ for $t = t_1+1, \ldots t_2$

1: Initialize $err_{acc}(\lambda^{t_1}) \leftarrow 0$
2: Initialize $P \leftarrow \mathbf{U}_k^{t_1}\mathbf{U}_k^{t_1\prime}$
3: **for** $t = t_1+1$ to t_2 **do**
4: Calculate $impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1}) \leftarrow \| P\Delta\mathbf{A}^t \|_{Fro}$
5: Calculate $err_{acc}(\lambda^t) \leftarrow err_{acc}(\lambda^{t-1}) + impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1})$
6: **end for**
7: Return $err_{acc}(\lambda^t)$ for $t = t_1+1, \ldots t_2$

In Algorithm 4, $err_{acc}(\lambda^{t_1})$ is initialized as 0 in step 1 and P is initialized as the projection matrix in step 2. From step 3 to 6, the impact of each perturbation matrix is calculated and accumulated to $err_{acc}(\lambda^t)$. In step 7, the estimated error array $err_{acc}(\lambda^t)$ for $t = t_1+1, \ldots t_2$ is returned.

4.5.1. Complexity Analysis

The complexity of initializing projection matrix P is $O(n^2k)$. Since $\Delta\mathbf{A}^t$ is often very sparse, the complexity of calculating $impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1})$ can be reduced to $O(ns)$ where s is the number of changed edges at current time stamp. The complexity of accumulating $err_{acc}(\lambda^t)$ at each time stamp is $O(1)$. Therefore the overall time complexity for error estimation over time series of length T is $O(n^2k + Tns)$.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate TRIP-BASIC and TRIP on real datasets. All the experiments are designed to answer the following two questions

- *Effectiveness*: how accurate are our algorithms in tracking eigen-functions, analyzing corresponding attributions and estimating the tracking errors?
- *Efficiency*: how fast are the tracking algorithms?

5.1. Experiment Setup

Machine.
We ran our experiment in a machine with two Intel Xeon 3.5GHz processors with 256GB of RAM. Our experiment is implemented with Matlab using single thread.
Datesets.
AS The first dataset we use for the evaluation is Autonomous system graph, which is available at http://snap.stanford.edu/data/. The graph has recorded communications between routers in the Internet for a long period of time. Based on the data from [61], we constructed an undirected dynamic communication graph that contains 100 daily instances with time span from November 8, 1997 to February 16, 1998. The largest graph among those instances has 3569 nodes and 12,510 edges. The dataset shows both the addition and deletion of nodes and edges over time.
Power Grid The second dataset is power grid network. It is a static, undirected, unweighted network representing the topology of the Western States Power Grid of the United States [62], which has 4941 nodes and 6594 edges. To simulate the evolving process, we randomly add 0.5%m (m is the number of edges in the graph) new edges to the graph at each time stamp as perturbation edges. We have changed different percentages of perturbation edges, and experimented several runs on each of the settings. As the results are similar, we only report the results from one run for brevity.
Airport The third dataset is a static, undirected, unweighted airport network, which represents the internal US air traffic lines between 2649 airports and has 13,106 links (available at http://www.levmuchnik.net/Content/Networks/NetworkData.html). Again, similar synthetic evolving process was done on this dataset. With similar experiment results, we only report those from one run of simulation for brevity.
Evaluation Metrics.
For the quality of eigen-functions tracking, we use the error rate ϵ. For eigenvalues, number of triangles and robustness measurement, their error rate are computed as $\epsilon = \frac{|\mathrm{f}-\mathrm{f}^*|}{\mathrm{f}^*}$, where f and f* are the estimated and true eigen-function values, respectively. For eigenvector, the error is computed as $\epsilon = 1 - \frac{\mathbf{u}\mathbf{u}^*}{\|\mathbf{u}\|\|\mathbf{u}^*\|}$, where $\mathbf{u}$ is the estimated eigenvector and $\mathbf{u}^*$ is the corresponding true eigenvector. For attribution analysis, we use the top-10 precision. For efficiency, we report the speedup of our algorithms over the

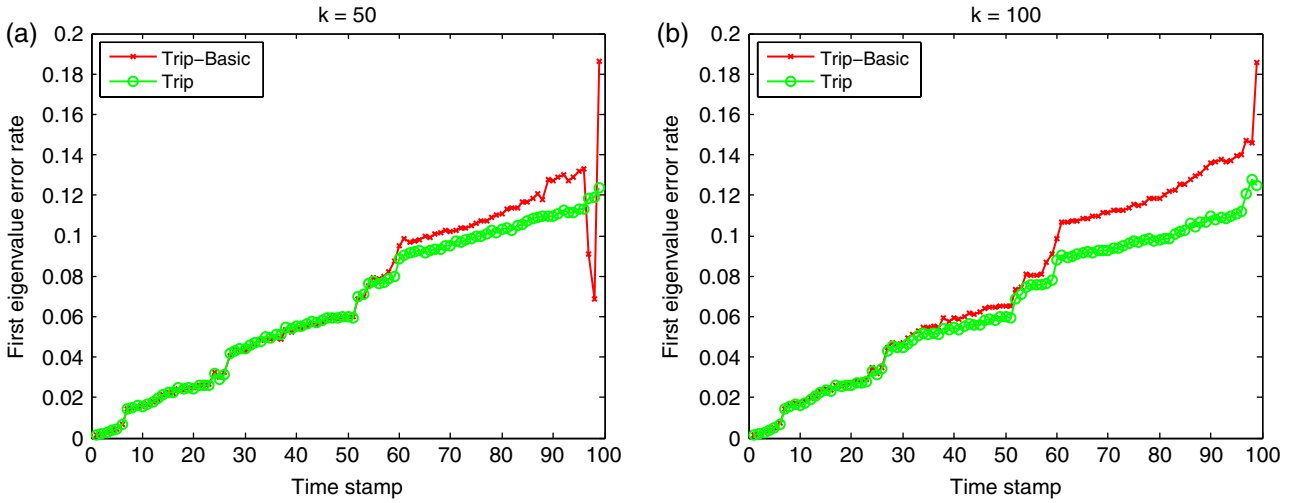

Fig. 2 The error rate of first eigenvalue approximation (a) $k = 50$ and (b) $k = 100$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

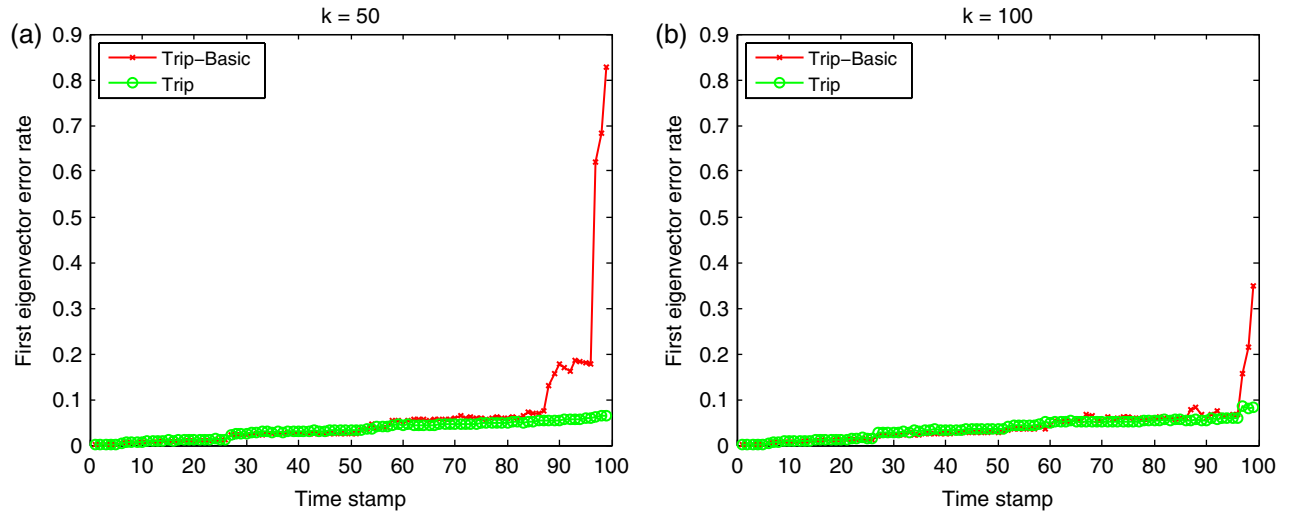

Fig. 3 The error rate of first eigenvector approximation (a) $k = 50$ and (b) $k = 100$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

re-computing strategy which computes the corresponding eigen-pairs from scratch at each time stamp.

5.2. Effectiveness Results

A. Effectiveness of Eigen-Function Tracking. Figs. 2–6 compare the effectiveness of TRIP-BASIC and TRIP using different number of eigen-pairs (k). We have the following observations. First, for all of the four eigen-functions, both algorithms could reach an overall error rate below 20% at the end of the tracking process. Second, when k is increased from 50 to 100, TRIP-BASIC could get a relatively more stable approximation over the tracking process. Third, TRIP is more stable and overall reaches a smaller error rate compared with TRIP-BASIC. For example, as time goes by, TRIP-BASIC starts to fluctuate sharply when $k = 50$ on all four eigen-functions. Finally, the error on the number of triangles is relatively higher. This is probably because that the number of triangles is the sum of cubic eigenvalues, and small errors on eigenvalues would therefore be magnified on the final result.

In addition, we also compared our algorithms with three different eigen-pair estimation methods, which include (1) "QR Decom", a QR decomposition-based eigen-pairs updating method [32]; (2) "SVD delta", simple SVD decomposition on $\Delta\mathbf{A}$; and (3) "Nystrom", a sampling-based eigen-pair estimation method derived from Nystrom algorithm [63]. For better effectiveness/efficiency trade-off,

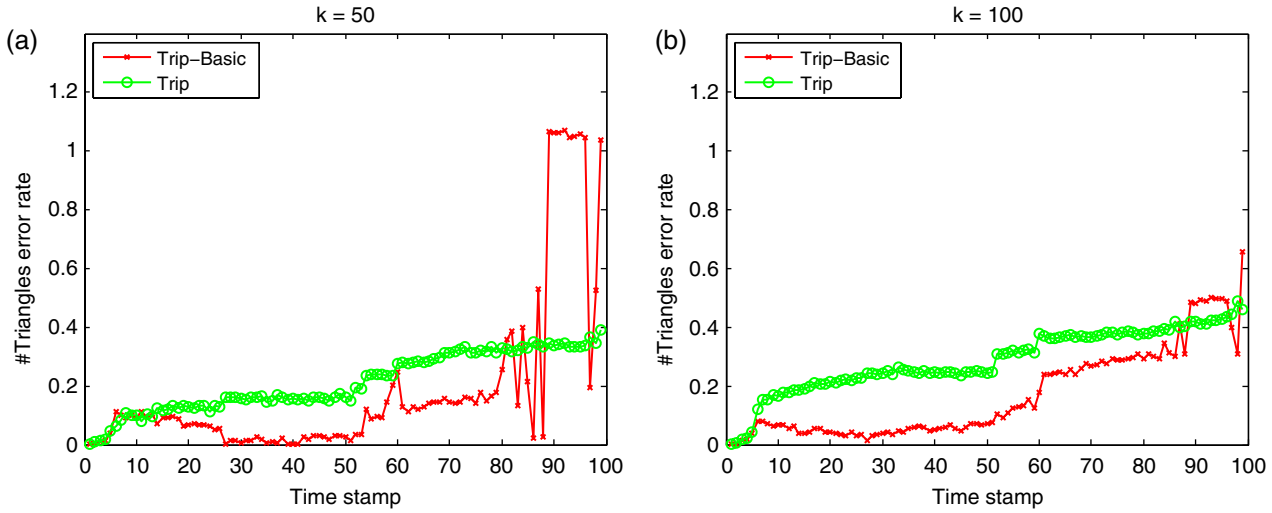

Fig. 4 The error rate of number of triangles approximation (a) $k = 50$ and (b) $k = 100$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

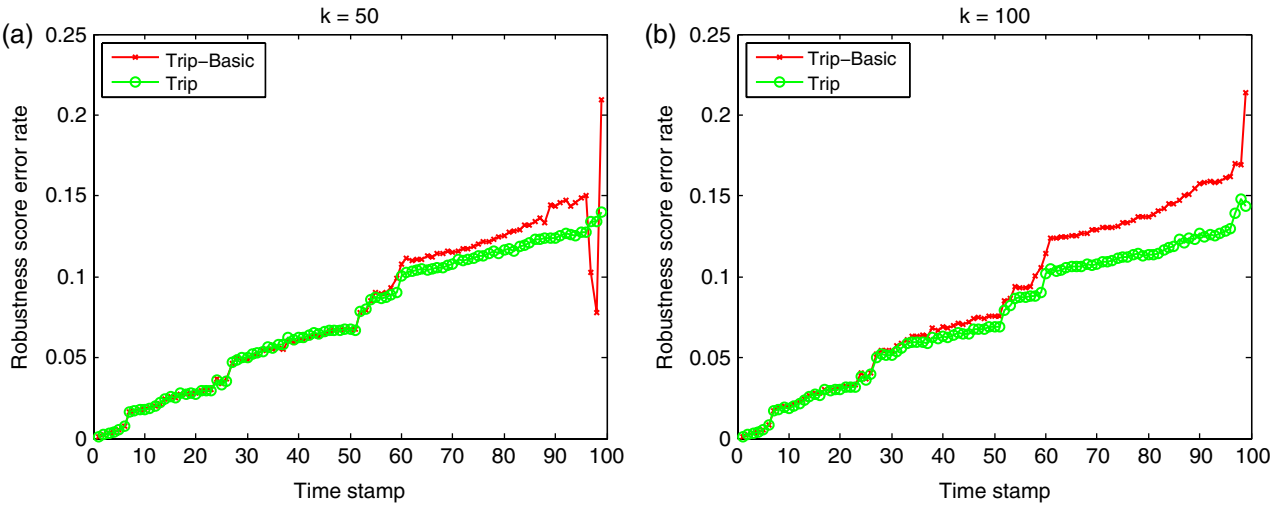

Fig. 5 The error rate of robustness score approximation (a) $k = 50$ and (b) $k = 100$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

we sample 2000 nodes for Nystrom algorithm to calculate eigen-pairs in our experiment. To better illustrate the results, we take the error rates of all methods for every 15 days on the *AS* data set. As "SVD delta" method causes large tracking errors compared to other methods, we only report the error rates from other comparing methods as shown from Figs 7–11. We can see that the performance of TRIP-BASIC and TRIP are among the best methods though their error rates keeps increasing as time accumulates.

B. Effectiveness of Attribution Analysis. For attribution analysis, we divided the changed edges at each time stamp into two classes: edges being added and edges being removed. Also among these two classes, we rank those edges according to their attribution score defined in Section 4. As a consequence, the top ranked edges are the ones that have most impact on the corresponding eigen-functions. Here we scored and ranked those edges with our approximated eigen-pairs and true eigen-pairs, respectively, and then compare the similarity between the two ranks. The precision of attribution analysis therefore is defined as the precision at rank 10 in approximated rank list. As similar results are observed in all three data set, we only report those on *AS* dataset as shown in Figs. 12 and 13. For the analysis on both added edges and removed edges, TRIP overall outperforms TRIP-BASIC.

C. Effectiveness of Error Estimation. To show the effectiveness of Algorithm 4, we compare the curve shapes between true errors of TRIP and accumulative estimated

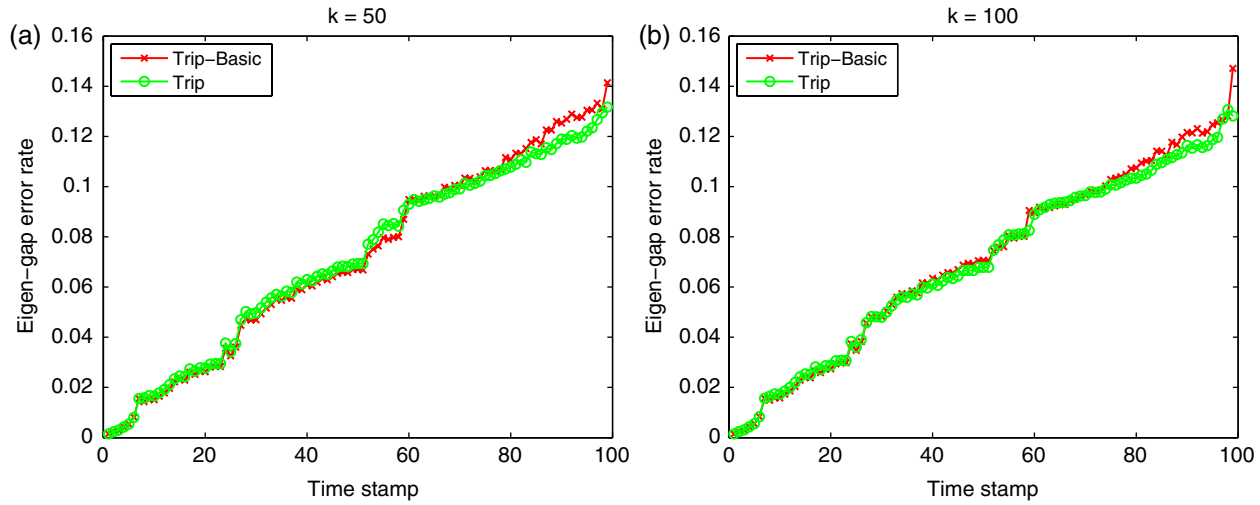

Fig. 6 The error rate of eigen-gap approximation (a) $k = 50$ and (b) $k = 100$. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

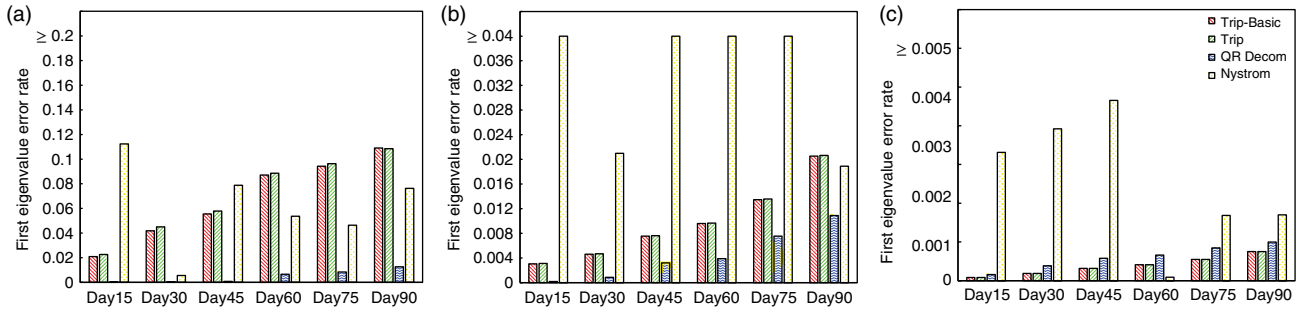

Fig. 7 The error rate of first eigenvalue approximation (a) AS, (b) Power Grid and (c) Airport. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

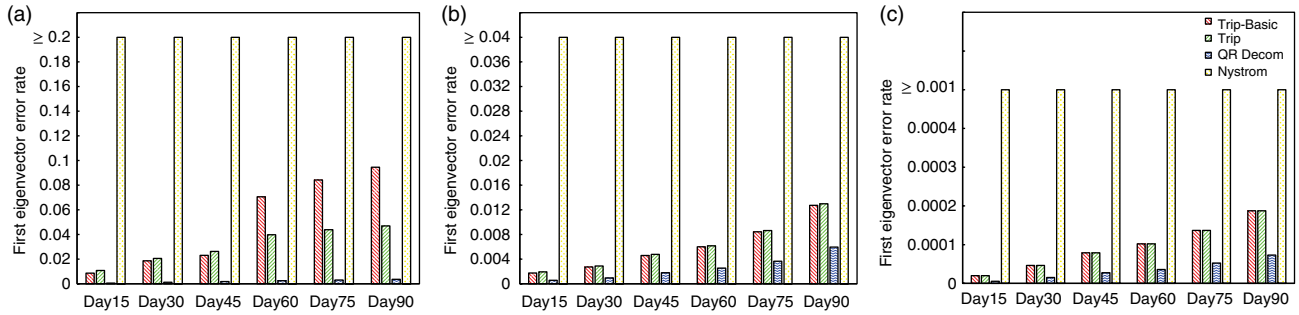

Fig. 8 The error rate of first eigenvector approximation (a) AS, (b) Power Grid, and (c) Airport. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

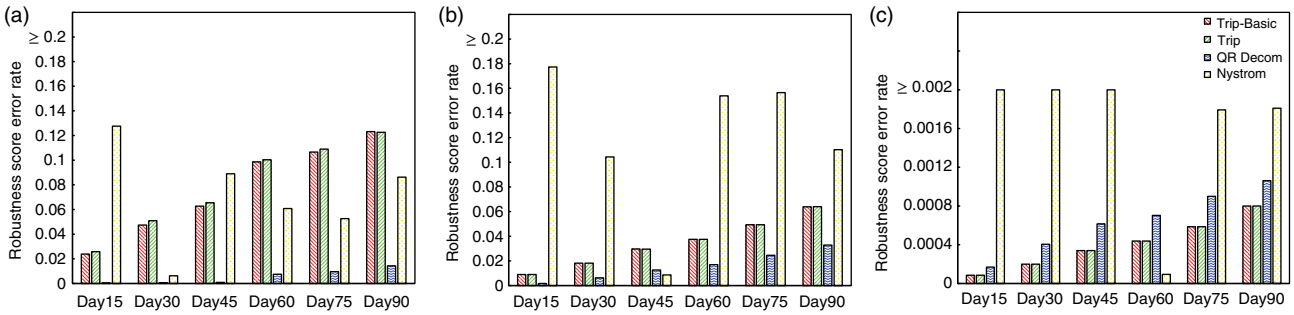

Fig. 9 The error rate of robustness score approximation (a) AS, (b) Power Grid, and (c) Airport. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

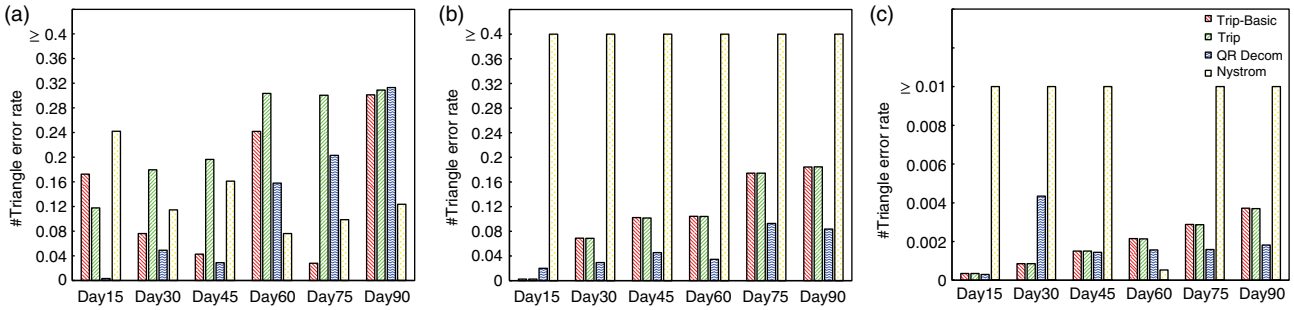

Fig. 10 The error rate of number of triangles approximation (a) AS, (b) Power Grid, and (c) Airport. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

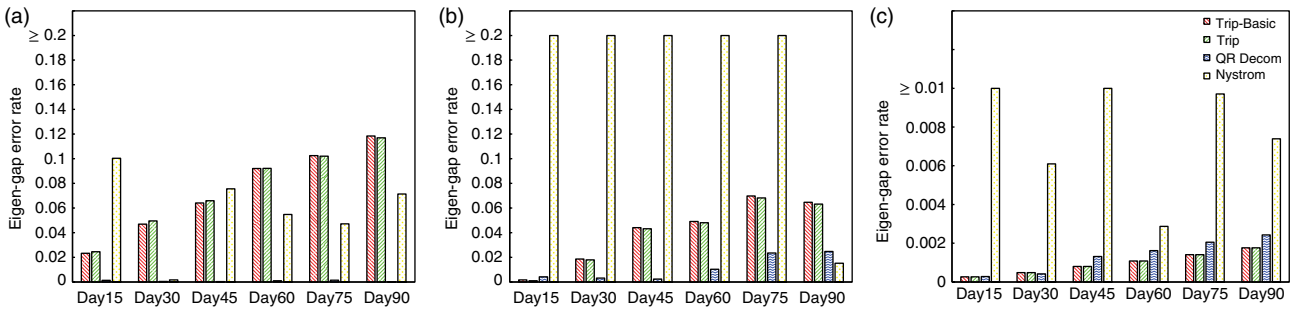

Fig. 11 The error rate of eigen-gap approximation (a) AS, (b) Power Grid, and (c) Airport. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

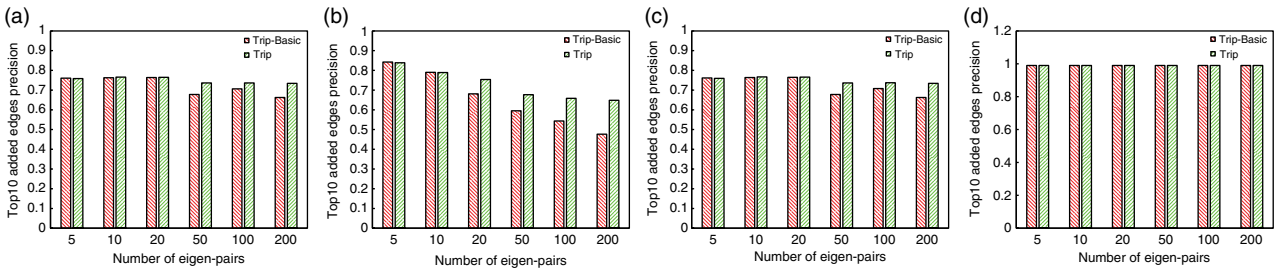

Fig. 12 Average precision over time for the attribution analysis (added edges) (a) First Eigenvalue, (b) Number of Triangles, (c) Robustness and (d) Eigen-Gap. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

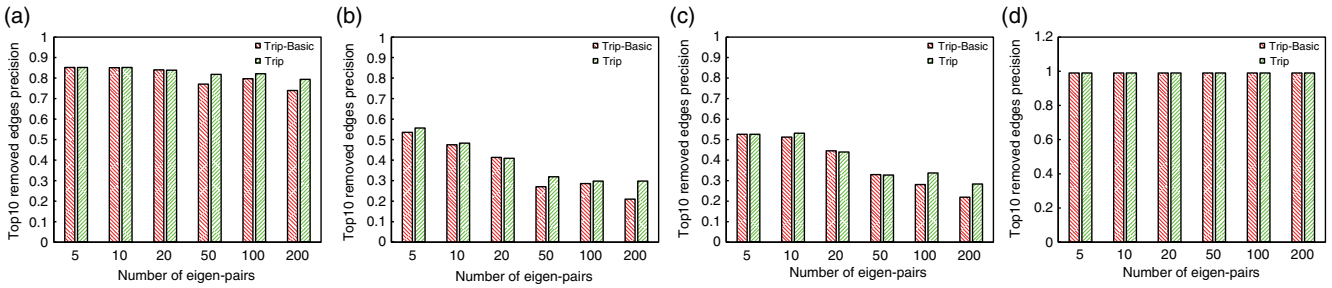

Fig. 13 Average precision over time of the attribution analysis (removed edges) (a) First Eigenvalue, (b) Number of Triangles, (c) Robustness and (d) Eigen-Gap. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

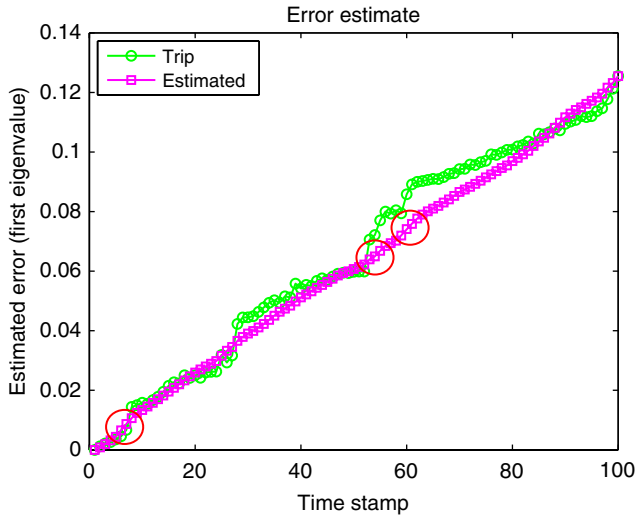

Fig. 14 The estimated error of TRIP-BASIC and TRIP on **AS** data set. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

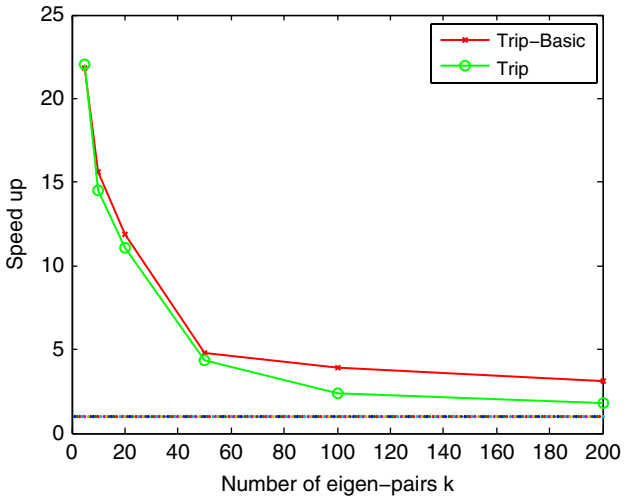

Fig. 15 The running time speedup of TRIP-BASIC and TRIP wrt to k. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

errors $err_{acc}(\lambda^t)$ on *AS* data set with $k = 50$. Ideally, the two curves should overlap with each other when $err_{acc}(\lambda^t)$ is properly scaled with some elaborately picked factor. Fig. 14 shows that the estimated error $err_{acc}(\lambda^t)$ can effectively catch sharp error increases in the tracking process as marked in red circle. Therefore, it can be used as a trigger to re-start the tracking process so that the accumulative error can always be kept within a low range.

5.3. Efficiency Results

Fig. 15 shows the average speed up with respect to different k values on *AS* dataset. We see that both TRIP-BASIC and TRIP can achieve more than 20× speed up when k is small. As k increases, the speedup decreases.

To further demonstrate the efficiency of the proposed algorithms, we also compare their effectiveness/efficiency trade-offs with those of the alternative methods mentioned in the previous subsection. Fig. 16 shows that our algorithms can keep the average error rate very small on all three data sets while consuming least amount of time.

6. CONCLUSION

In this paper, we study the problem of eigen-functions tracking on dynamic graphs. We first introduce different kinds of eigen-functions and their applications. In order to efficiently track these functions over time, we propose TRIP-BASIC and TRIP. In addition, we provide a framework for attribution analysis on eigen-functions and a method to effectively estimate tracking errors. Our experiments show that both TRIP-BASIC and TRIP can effectively and efficiently track the changes of eigen-pairs, number of triangles, robustness score and eigen-gap in dynamic graphs, while TRIP is more stable over time. In both cases, the accumulated error rate inevitably keeps increasing as time goes by.

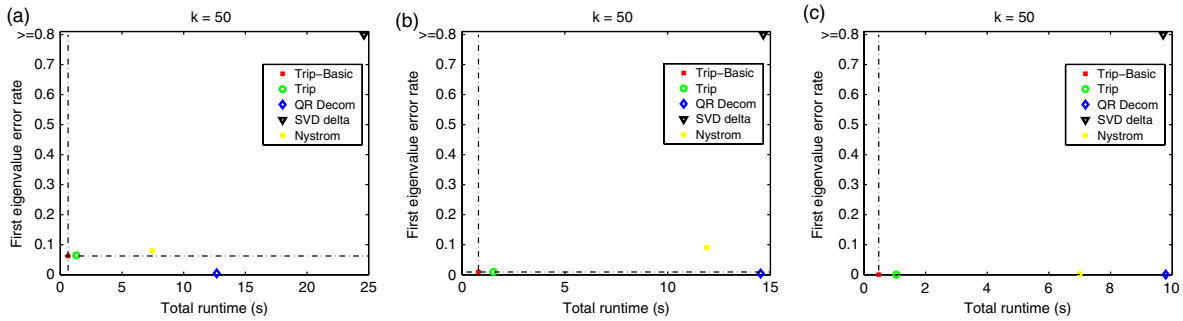

Fig. 16 The error rate vs. total runtime of first eigenvalue approximation in 100 time stamps (a) AS, (b) Power Grid and (c) Airport. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

ACKNOWLEDGEMENT

This material is supported by the National Science Foundation under Grant No. IIS1017415, by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, by Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0200 and W911NF-12-C-0028, by National Institutes of Health under the grant number R01LM011986, Region II University Transportation Center under the project number 49997-33 25.

The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] M. E. J. Newman, The mathematics of networks, New palgrave Encycl Econ 2 (2008), 1–12.
[2] D. Chakrabarti, Y. Wang, C. Wang, and J. Leskovec, C. Faloutsos, Epidemic thresholds in real networks, ACM Trans Inform Syst Secur 10 (4) (2008), 1.
[3] B. Aditya Prakash, D. Chakrabarti, N. C. Valler, M. Faloutsos, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks, Knowl Inf Sys 33 (3) (2012), 549–575.
[4] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In 22nd International Symposium on Reliable Distributed Systems, 2003 Proceedings, IEEE, 2003, 25–34.
[5] Stanley Wasserman, Social network analysis: methods and applications, 8, Cambridge University Press, 1994.
[6] R. Albert, H. Jeong, and A.-L. Barabasi, Error and attack tolerance of complex networks, Nature 406 (6794) (2000), 378–382.
[7] Hau Chan, Leman Akoglu, and Hanghang Tong. Make it or break it: manipulating robustness in large networks. In SDM, SIAM, 2014.
[8] H. Frank, and I. Frisch, Analysis and design of survivable networks, IEEE Trans Commun Technol 18 (5) (1970), 501–519.
[9] C. E. Tsourakakis. Fast counting of triangles in large real networks without counting: algorithms and laws. In Eighth IEEE International Conference on Data Mining, 2008. ICDM'08, IEEE, 2008, 608–617.
[10] J. Wu, B. Mauricio, Y.-J. Tan, and H.-Z. Deng, Natural connectivity of complex networks, Chin Phys Lett 27 (7) (2010), 78902.
[11] F. R. K. Chung. Spectral graph theory, Amn Math Soc 92 (1997).
[12] S. Hoory, N. Linial, and A. Wigderson, Expander graphs and their applications, Bull Am Math Soc 43 (4) (2006), 439–561.
[13] F. Harary and Allen Schwenk, The spectral approach to determining the number of walks in a graph, Pac J Math 80 (2) (1979), 443–449.
[14] C. E. Tsourakakis, Counting triangles in real-world networks using projections, Knowl Inf Sys 26 (3) (2011), 501–520.
[15] A. Ganesh, L. Massoulié, and D. Towsley, The effect of network topology on the spread of epidemics. In INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies 2 (2005), 1455–1466.
[16] H. Tong, B. Aditya Prakash, C. Tsourakakis, T. Eliassi-Rad, Christos Faloutsos, and D. H. Chau. On the vulnerability of large graphs. In 2010 IEEE 10th International Conference on Data Mining (ICDM), IEEE, 2010, 1091–1096.
[17] H. Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and melting, large graphs by edge manipulation. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management. ACM, 2012, 245–254.
[18] L. T. Le, T. Eliassi-Rad, and H. Tong. Met: a fast algorithm for minimizing propagation in large graphs with small eigen-gaps. In SDM, SIAM, 2015.
[19] E. Estrada, Network robustness to targeted attacks. The interplay of expansibility and degree distribution, Eur Phys J B 52 (4) (2006), 563–574.
[20] F. D. Malliaros, V. Megalooikonomou, and C. Faloutsos, Estimating robustness in large social graphs, Know Inf Sys 45 (2015), 645–678.
[21] Hau Chan, Shuchu Han, and Leman Akoglu, Where graph topology matters: the robust subgraph problem. In SDM. SIAM, 2015.
[22] M. E. J. Newman, Finding community structure in networks using the eigenvectors of matrices, Phys Rev E 74 (3) (2006), 036104.
[23] M. E. J. Newman, Modularity and community structure in networks, Proc Natl Acad Sci 103 (23) (2006), 8577–8582.

[24] C. Aggarwal, and K. Subbian, Evolutionary network analysis: a survey, ACM Comput Surv 47 (1) (2014), 10.
[25] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, ACM, 2005, 177–187.
[26] J. Leskovec, J. Kleinberg, and C. Faloutsos, Graph evolution: densification and shrinking diameters, ACM Trans Knowl Discov Data 1 (1) (2007), 2.
[27] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos, Fast monitoring proximity and centrality on time-evolving bipartite graphs, Stat Anal Data Min 1 (3) 2008, 142–156.
[28] F. D. Malliaros, V. Megalooikonomou, and C. Faloutsos, Fast robustness estimation in large social graphs: communities and anomaly detection. In SDM, SIAM 12 (2012), 942–953.
[29] T. Idé, and H. Kashima. Eigenspace-based anomaly detection in computer systems. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2004, 440–449.
[30] J. Ferlez, C. Faloutsos, J. Leskovec, D. Mladenic, and M. Grobelnik, Monitoring network evolution using mdl. In IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008, IEEE, 2008, 1328–1330.
[31] A. Barron, J. Rissanen, and B. Yu, The minimum description length principle in coding and modeling, IEEE Trans Inform Theory 44 (6) (1998), 2743–2760.
[32] L. Li, H. Tong, Y. Xiao, and W. Fan. Cheetah: fast graph kernel tracking on dynamic graphs. In SDM, SIAM, 2015.
[33] N. Huazhong, X. Wei, C. Yun, G. Yihong, and T. S. Huang, Incremental spectral clustering by efficiently updating the eigen-system, Pattern Recogn 43 (1) (2010), 113–127.
[34] R. Albert, H. Jeong, and A.-L. Barabasi, Diameter of the world wide web, Nature 401 (1999), 130–131.
[35] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, Graph structure in the web: experiments and models. In WWW Conference, 2000.
[36] M. Faloutsos, P. Faloutsos, and C. Faloutsos, On power-law relationships of the internet topology, In ACM SIGCOMM Computer Communication Review, 29 1999, 251–262.
[37] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal. Discovering frequent topological structures from graph datasets. In KDD, 2005, 606–611.
[38] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (5594) (2002), 824–827.
[39] D. Xin, J. Han, X. Yan, and H. Cheng, Mining compressed frequent-pattern sets. In VLDB, 2005, 709–720.
[40] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In KDD, 2006, 44–54.
[41] G. Karypis and V. Kumar. Multilevel -way hypergraph partitioning. In DAC, 1999, 343–348.
[42] F. Geerts, H. Mannila, and E. Terzi, Relational link-based ranking. In VLDB 2004, 552–563.
[43] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In ICDM, 2006, 613–622.
[44] H. Tong, J. He, M. Li, W.-Y. Ma, H.-J. Zhang, and C. Zhang. Manifold-ranking-based keyword propagation for image retrieval. EURASIP J Appl Signal Process 2006, 79412. doi:10.1155/ASP/2006/79412.
[45] A. S. Maiya, Sampling and inference in complex networks, PhD thesis, Stanford University, 2011.
[46] A. S. Maiya and T. Y. Berger-Wolf, Benefits of bias: towards better characterization of network sampling. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, 105–113.
[47] D. Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, Washington, DC, USA 2003, 137–146.
[48] N. C. Valler. Spreading Processes on Networks Theory and Applications. PhD thesis, University of California, 2012.
[49] X. Wei, N. C. Valler, B. Aditya Prakash, I. Neamtiu, M. Faloutsos, and C. Faloutsos, Competing memes propagation on networks: a network science perspective, IEEE J Selected Areas in Commun 31 (6) (2013), 1049–1060.
[50] S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin, Catastrophic cascade of failures in interdependent networks, Nature 464 (7291) (2010), 1025–1028.
[51] A. Vespignani, Complex networks: the fragility of interdependency. Nature 464 (7291) 2010, 984–985.
[52] J. Gao, S. V Buldyrev, H. E. Stanley, and S. Havlin, Networks formed from interdependent networks, Nat Phys 8 (1) (2012), 40–48.
[53] R. Parshani, S. V. Buldyrev, and S. Havlin, Interdependent networks: reducing the coupling strength leads to a change from a first to second order percolation transition, Phys Rev Lett 105 (4) (2010), 048701.
[54] A. Sen, A. Mazumder, J. Banerjee, A. Das, and R. Compton, Multi-layered network using a new model of interdependency, arXiv preprint arXiv 1401.1783, 2014.
[55] J. Shao, S. V. Buldyrev, S. Havlin, and H. E. Stanley, Cascade of failures in coupled network systems with multiple support-dependent relations. arXiv preprint arXiv:1011.0234, 2010.
[56] C. Chen, J. He, N. Bliss, and H. Tong. On the connectivity of multi-layered networks: models, measures and optimal control. In ICDM. IEEE, 2015.
[57] J. Gao, S. V. Buldyrev, S. Havlin, and H. E. Stanley, Robustness of a network of networks, Phys Rev Lett 107 (19) (2011), 195701.
[58] B. Aditya Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. Faloutsos. Eigenspokes: surprising patterns and scalable community chipping in large graphs. In Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Springer Berlin Heidelberg, Hyderabad, India, June 21–24, 2010. Proceedings. Part II, 2010, 435–448.
[59] V. V. Williams, Breaking the coppersmith-winograd barrier.
[60] G. W. Stewart, and S. Ji-Guang, Matrix Perturbation Theory, Academic Press, 1990.
[61] University of Oregon Route View Project. Online data and reports. http://www.routeviews.org.
[62] J. Duncan, Watts and Steven H Strogatz. Collective dynamics of small-world networks, Nature 393 (6684) (1998), 440–442.
[63] P. Drineas, and M. W. Mahoney, On the nyström method for approximating a gram matrix for improved kernel-based learning, J Mach Learn Res 6 (2005), 2153–2175.