Fast Eigen-Functions Tracking on Dynamic Graphs

Chen Chen
Arizona State University
cchen211@asu.edu

Hanghang Tong
Arizona State University
hanghang.tong@asu.edu

Abstract

Many important graph parameters can be expressed as eigen-functions of its adjacency matrix. Examples include epidemic threshold, graph robustness, etc. It is often of key importance to accurately monitor these parameters. For example, knowing that Ebola virus has already been brought to the US continent, to avoid the virus from spreading away, it is important to know which emerging connections among related people would cause great reduction on the epidemic threshold of the network. However, most, if not all, of the existing algorithms computing these measures assume that the input graph is static, despite the fact that almost all real graphs are evolving over time. In this paper, we propose two online algorithms to track the eigen-functions of a dynamic graph with linear complexity wrt the number of nodes and number of changed edges in the graph. The key idea is to leverage matrix perturbation theory to efficiently update the top eigen-pairs of the underlying graph without re-computing them from scratch at each time stamp. Experiment results demonstrate that our methods can reach up to $20\times$ speedup with precision more than 80% for fairly long period of time.

Keywords: dynamic graph; connectivity; graph spectrum; attribution analysis

1 Introduction

Among others, node importance and graph connectivity are of key importance to understand fundamental characteristics of a network (such as social networks, power grid, transportation network, etc). To date, many different parameters of the graph have been invented to measure those properties from different perspective. One most commonly used parameter for node importance estimation is eigenvector centrality [15], which is effective on identifying influential nodes over the whole network. As for connectivity in the graph, important parameters include epidemic threshold ([28, 4, 20]), clustering coefficient [29] and graph robustness ([2, 8, 5]). One interesting observation is that many of those parameters can be calculated or estimated accurately by certain *functions of the eigen-pairs* of the graph. For example, it has been found that for an arbitrary graph, the tipping point for the dissemination process is controlled by the leading eigenvalue of certain system matrix associated with the graph [28, 20]. As for the clustering coefficient computation, instead of doing $\binom{n}{3}$ inspections for counting triangles, [26] proposed a fast counting algorithm based on the top eigenvalues of the graph. Moreover in [5], Chan et al. showed that natural connectivity [31], a function based on top eigenvalues of the graph, is a good measurement of graph robustness.

Most of the algorithms that compute the above parameters/eigen-functions are based on static graphs. However in real world networks, the graph structure keeps evolving over time. It is of great importance to keep track of those measurements since subtle changes on graph structure may lead to sharp changes on the overall properties. For example, in epidemic process, some emerging connections between people may increase the leading eigenvalue a lot, and thus reduce the epidemic threshold, which in turn makes the virus easier to spread through the network. By monitoring related parameters in the network over time, we would be able to know when the change happens and identify its cause/attribution timely. Consider another scenario in social network sites, since the connections between users could change dramatically day by day, the influential individuals would therefore be changed as well, which is important for online marketing companies change their ads targeting strategy over time.

However, simply re-computing the eigen-pairs whenever the graph has been changed is computationally costly if not infeasible over fast changing large graphs. The popular Lanczos method would require $O(mk + nk^2)$ time to compute the top-k eigen-pairs, where m and n are the numbers of edges and nodes in the graph. Such a complexity, even though might be acceptable for static graphs, would be expensive for dynamic graphs. To address this chalellenge, instead of re-computing the eigen-pairs from scratch at each time stamp, we consider to update the eigen-pairs based on its previous ones. In this paper, we propose two online algorithms to track the eigen-pairs of a dynamic graph efficiently. Our algorithms have a linear time complexity wrt the number of nodes n in the graph and the number of changed edges s in each time stamp, where s is often much smaller than the total number of the edges m.

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. At each time stamp, we provide additional methods for attribution analysis on eigen-functions.
- *Evaluations*. We evaluate our methods on real-world dataset, demonstrating their effectiveness, efficiency and the balance between these two.

The rest of the paper is organized as follows: In Section 2, a brief survey of related studies on graph spectrum and dynamic graphs are provided. A formal problem definition is given in Section 3. Section 4 gives the first order and high order eigen-functions tracking algorithms 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 many different important parameters 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 two sections: (A) work on applications of different eigen-functions, and (B) work on dynamic graph analysis.

2.1 Applications of Eigen-functions

According to [6], 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 [10], 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 [26] and [27], 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 [9] and [4], 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 is depended on the first eigenvalue of certain system matrix associated with the network. Tong et al. proposed a node manipulation method in [25] and edge manipulation method in [24] to optimize the change of first eigenvalue in the graph. In [5], Chan et al. proposed a more general robustness measurement and provided corresponding graph manipulating strategies (on both nodes and edges) to optimize the robustness score. On the other hand for Laplacian matrix of the graph, Newman has shown that the eigen-pairs of Laplacian matrix can be used for community detection [16],[17]. 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 [1]. The research on dynmaic graph analysis can be generally sorted into two categories: (A) monitoring the change on the evolving graph and (B) efficiently update the data mining results over graph changes.

In [12] and [13], Leskovec et al. discovered the growth pattern of real graphs by their densities and diameters. As graph mining tasks varies from one another, the parameters tracked in the process are different. In [23], two online algorithms were provided for tracking node proximity and centrality on bipartite graphs. In [14], Malliaros et al. 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 [11] based on eigen-pairs of dependency matrix of the graph. Ferlez et al. [7] proposed a dynamic graph monitoring algorithm based on MDL [3] which can be used to detect the changing communities in the evolving process. The other area of research that is remotely related to our work is evolutionary spectral clustering on graphs. In [18], Ning et al. proposed an incremental spectral clustering algorithm based on iterative update on the eigen-system of the graph.

3 Problem Definition

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

3.1 Notations The symbols used through out 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 is 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

$$\text{f} : (\Lambda_k, \mathbf{U}_k) \to \mathbb{R}^x (x \in \mathbb{N}) \tag{3.1}$$

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
$({\lambda_j}^t, {\mathbf{u_j}}^t)$	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

3.2 Important Eigen-Functions .

Eigenvalues and Eigenvectors Since the eigen-pairs of a graph are important attibutes themselves, the simpliest eigen-function is therefore an identity function as follows.

$$f((\Lambda_k, \mathbf{U}_k)) = (\Lambda_k, \mathbf{U}_k) \tag{3.2}$$

The eigenvalues of a graph's adjacency matrix can be used to measure the path capacity of a graph [10], while the eigenvectors can be used to evaluate the centrality of nodes [15], or to detect interesting subgraphs [21].

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.

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 reduce the complexity to $O(n^{2.373})$ [30], but this is still not a scalable algorithm on real world large dataset. In [26], Tsourakakis proposed a fast triangle counting algorithm which showed that the number of triangles in a graph($\triangle(G)$) can be estimated using Equ. (3.3).

$$f((\Lambda_k, \mathbf{U}_k)) = \triangle(G) = \frac{1}{6}\sum_{i=1}^{k}\lambda_i^3 \tag{3.3}$$

By Equ. (3.3), number of triangles $\triangle(G)$ therefore becomes a function of eigenvalues Λ_k. Again, for real-world graphs, we usually only need top k eigenvalues to achieve a good approximation for triangle counting. For example, experiments in [26] showed that picking top 30 eigen-pairs can achieve an accuracy of at least 95% in most graphs.

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. [5] 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. The definition of robustness score($S(G)$) [5] is shown in Equ. (3.4).

$$f((\Lambda_k, \mathbf{U}_k)) = S(G) = \ln(\frac{1}{k}\sum_{j=1}^{k}e^{\lambda_j}) \tag{3.4}$$

By Equ. (3.4), robustness score $S(G)$ is also a function of eigenvalues Λ_k.

Once again, [5] found that top k ($k = 50$ in their study) eigen-pairs are sufficient for estimating robustness score.

3.3 Problem Definition

In all the above cases, the network parameters of interest (e.g., epidemic threshold, eigen centrality, the number of triangles, the robustness measurement) 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 (3.2) to (3.4) 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 for different eigen-functions.

4.1 TRIP-BASIC

Given any pair of adjacency matrices of a dynamic graph G in two consecutive time stamps, $\mathbf{A}^t$ and $\mathbf{A}^{t+1}$, the adjacency matrix at $t+1$ can be viewed as a perturbated version of that at t, with the perturbation as $\Delta\mathbf{A}^t = \mathbf{A}^{t+1} - \mathbf{A}^t$. By (first order) matrix perturbation theory [22], we can approximate the eigen-pairs of $\mathbf{A}^{t+1}$ by that of $\mathbf{A}^t$ using Equ. (4.5), without re-computing them from scratch.

$$\lambda_j^{t+1} = \lambda_j^t + \Delta\lambda_j = \lambda_j^t + \mathbf{u_j}^{t\prime}\Delta\mathbf{A}\mathbf{u_j}^t$$

(4.5)

$$\mathbf{u_j}^{t+1} = \mathbf{u_j}^t + \Delta\mathbf{u_j} = \mathbf{u_j}^t + \sum_{i=1,i\neq j}^{k}\left(\frac{\mathbf{u_i}^{t\prime}\Delta\mathbf{A}\mathbf{u_j}^t}{\lambda_j^t - \lambda_i^t}\mathbf{u_i}^t\right)$$

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 Equ. (4.5), the term $\mathbf{u_j}^{t\prime}\Delta\mathbf{A}\mathbf{u_j}^t$ can be expanded as

$$\mathbf{u_j}^{t\prime}\Delta\mathbf{A}\mathbf{u_j}^t = \sum_{<p,r>\in\Delta E}\Delta\mathbf{A}(p,r)\mathbf{u_{pj}}^t\mathbf{u_{rj}}^t \tag{4.6}$$

Equ. (4.5) and Equ. (4.6) naturally lead to our base solution (TRIP-BASIC) for solving Problem 1 as follows.

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:             $\Delta\mathbf{u_j} \leftarrow \Delta\mathbf{u_j} + \frac{\mathbf{u_i^t}'\Delta\mathbf{A^t}\mathbf{u_j^t}}{\lambda_j^t - \lambda_i^t}\mathbf{u_i^t}$
6:         end for
7:         Calculate $\Delta\lambda_j \leftarrow \mathbf{u_j^t}'\Delta\mathbf{A^t}\mathbf{u_j^t}$
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 $[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1+1\ldots t_2}$
```

The approximated eigen-pairs for each time stamp is computed from step 2 to 10. Each $\Delta\lambda_j$ and $\Delta\mathbf{u}_j$ is calculated from step 3 to 7 by Equ. (4.5) and (4.6). At step 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.

Complexity Analysis The efficiency of proposed Algorithm 1 is summarized in Lemma 4.1. Both time complexity and space complexity is linear wrt the total number of the nodes in the graph (n) and total number of the time stamps (T).

LEMMA 4.1. **Complexity of First Order Eigen-Function Tracking.** *Suppose T is the total number of the time 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 Alg. 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 Equ. (4.6), the complexity of computing term $\mathbf{u_j}^{t\prime}\Delta\mathbf{A}\mathbf{u_j}^t$ is $O(s)$, so the overall complexity of step 5 is $O(s+n)$. Therefore calculating $\Delta\mathbf{u_j}$ from step 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 $\mathbf{U_k}^t$ for each time stamp. In 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 update phase can be reused in each iteration. Therefore the overall space complexity for T time stamps takes a space of $O(Tnk+s)$.

4.2 TRIP

The baseline solution in Algorithm 1 is simple and straightforward, but it has the following limitations. First, the approximation error of first order matrix perturbation is in the order of $\|\Delta\mathbf{A}\|$. In other words, the quality of such an approximation might decreases quickly wrt the increase of $\|\Delta\mathbf{A}\|$. Second, the approximation quality is highly sensive to the small eigen-gap of $\mathbf{A}^t$ as indicated by Equ. (4.5). In order to address these limintations, 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 into consideration while updating eigenvectors. Details are shown in Appendix.

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 Appendix. Again, after we update the eigenvector in step 9, we normalize each of them to unit length.

Complexity Analysis The efficiency of Algorithm 2 is given in Lemma 4.2. Compared with TRIP-BASIC, both time and space complexity are still linear wrt total number of nodes in the graph and total number of time stamps, with a slight increase in k, which is often very small.

[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.

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 $[\Delta \mathbf{A}^t]_{t=t_1,\dots t_2-1}$
Output: Corresponding eigen-pairs $[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1+1,\dots t_2}$
1: for $t = t_1$ to $t_2 - 1$ do
2:     Calculate $\mathbf{X}^t \leftarrow \mathbf{U}_k^{t\,\prime} \Delta \mathbf{A}^t \mathbf{U}_k^t$
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 $p = 1, \dots, k$
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 $[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1+1\dots t_2}$
```

LEMMA 4.2. **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 $[\Delta\mathbf{A^t}]_{t=t_1,\dots t_2-1}$, then the time cost for Alg. 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 Equ. (4.6), calculating $\mathbf{X}^t$ in step 2 takes $O(k^2 s)$. Updating top eigenvalues in step 3-4 takes $O(k)$. From step 5-11, 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 $O(Tnk + k^2 + s)$.

Again, for the detailed derivation of TRIP, please refer to the Appendix.

4.3 Attribution Analysis

Based on our TRIP algorithm, 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 graph. 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.

5 Experimental Evaluation

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

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

5.1 Experiment Setup

Machine. We ran our experiment in a machine with 2 processors Intel Xeon 3.5GHz with 256GB of RAM. Our experiment is implemented with Matlab.

Dataset. The dataset we used for 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 [19], we constructed an undirected dynamic communication graph that contains 100 daily instances with time span from Nov 8 1997 to Feb 16 1998. The largest graph among those instances has 3,569 nodes and 12,510 edges. The dataset shows both the addition and deletion of nodes and edges over time.

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 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. Fig. 1 to 4 compare the effectiveness of TRIP-BASIC and TRIP using different number of eigen-pairs (k). We have the following observations. First, for all 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.

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. And among these two classes, we ranks 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. The results are shown in Fig. 5 and 6. For the analysis on both added edges and removed edges, TRIP overall outperforms TRIP-BASIC.

5.3 Efficiency Results

Fig. 7 shows the average speed up with respect to different k values. We see that both TRIP-BASIC and TRIP can achieve more than $20\times$ speed up when k is small. As k increases, the speedup decreases.

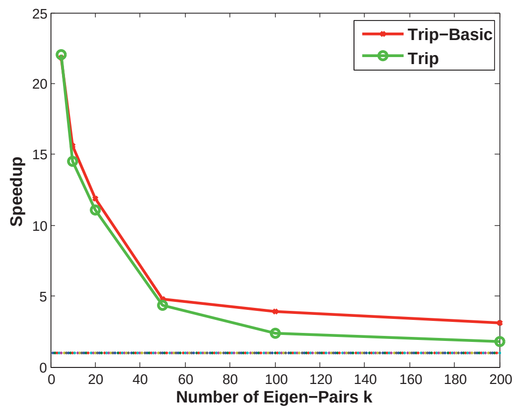


Figure 7: The running time speedup of TRIP-BASIC and TRIP wrt to k.

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 also provide a method to identify the attribution for the change of eigen-functions over time. Our experiments show that both TRIP-BASIC and TRIP can effectively and efficiently track the change of eigen-pairs, number of triangles and robustness score in the graph, while TRIP is more stable over time. In both cases, the accumulated error rate inevitably keeps increasing as time goes by. One interesting future research is to study when to reset the algorithms over time.

7 Appendix

Here, we provide the detailed derivations of eigen-pairs' update process used in TRIP. We would like to emphasize that the derivations themselves are based on the standard matrix perturbation theory [22], which are *not* the primary contribution of this paper.

For eigenvector approximation in TRIP, suppose $\Delta\mathbf{u_j} = \sum_{i=1}^{k} \alpha_{ij}\mathbf{u_i}^t$, let $\mathbf{X}^t(p,i)$ denotes $\mathbf{u_p}^{t'}\Delta\mathbf{A}\mathbf{u_i}^t(p,i = 1,\dots,k)$ and $\mathbf{D}^t$ denotes $diag(\lambda_j^t + \Delta\lambda_j - \lambda_p^t)$ for $p = 1,\dots,k$, we will show in the following that $\alpha_\mathbf{j} = (\mathbf{D}^t - \mathbf{X}^t)^{-1}\mathbf{X}^t(:,j)$ where $\alpha_\mathbf{j} = [\alpha_{1j},\dots,\alpha_{kj}]$

Give perturbation matrix $\Delta\mathbf{A}^t$ of $\mathbf{A}^t$, we have

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

Expanding the equation above, we get

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

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

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

Replacing all $\Delta\mathbf{u_j}$ terms with $\sum_{i=1}^{k}\alpha_{ij}\mathbf{u_i}$ and multiplying the term $\mathbf{u_p}^{t'}$ (for $1 \le p \le k$, $p \ne j$) on both size, by applying the orthogonality property of eigenvectors to the new equation, we have

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

which can be rewritten as

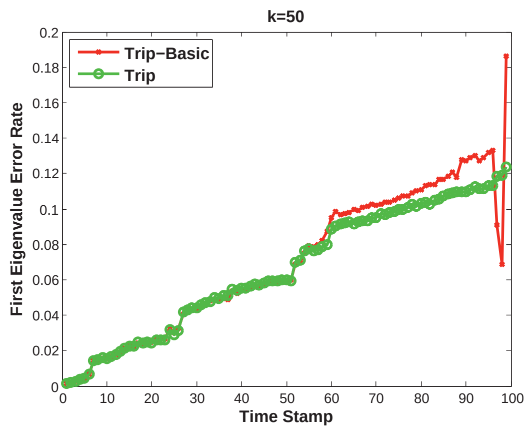
$$\mathbf{X}^t(p,j) - \alpha_{pj}(\lambda_j^t + \Delta\lambda_j - \lambda_p^t) + \sum_{i=1}^{k}\mathbf{X}^t(p,i)\alpha_{ij} = 0$$

By the definition of $\mathbf{X}^t$, $\mathbf{D}^t$ and $\alpha_\mathbf{j}$, the above equation can be expressed as

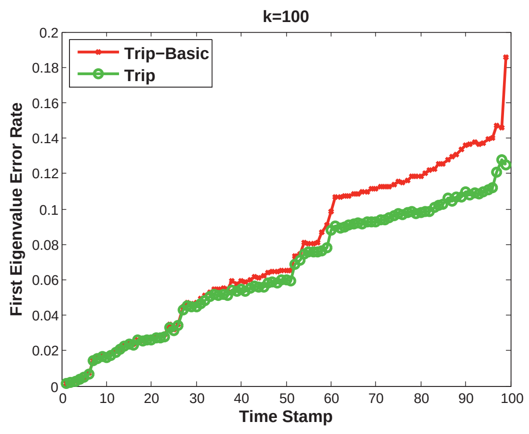
$$\mathbf{X}^t(:,j) - \mathbf{D}^t\alpha_\mathbf{j} + \mathbf{X}^t\alpha_\mathbf{j} = \mathbf{0}$$

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

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

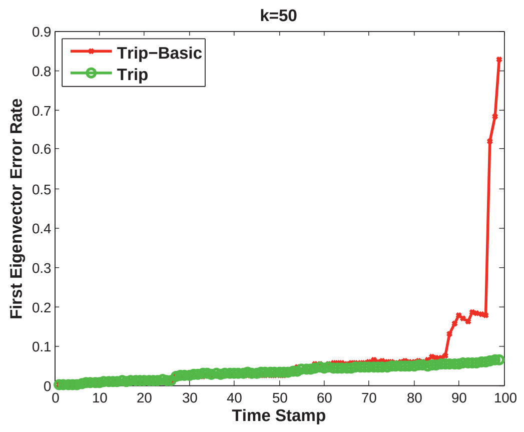


(a) k=50

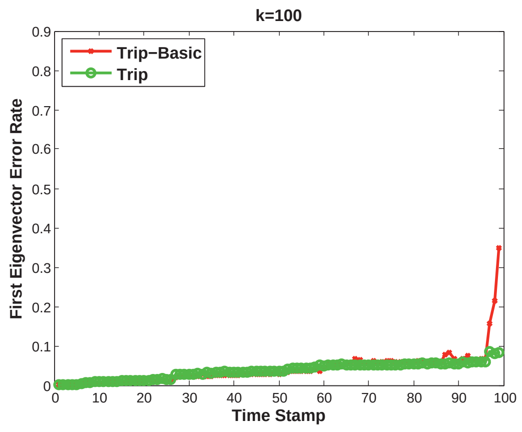


(b) k=100

Figure 1: The error rate of first eigenvalue approximation.

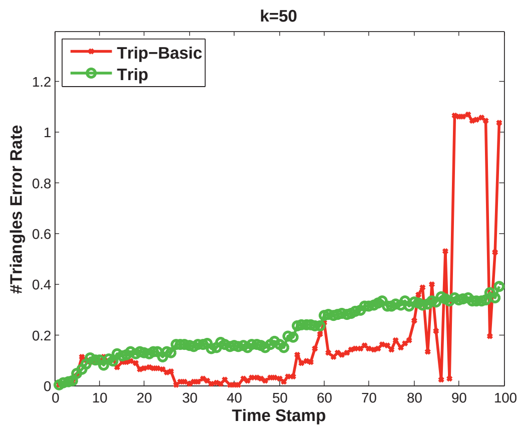


(a) k=50

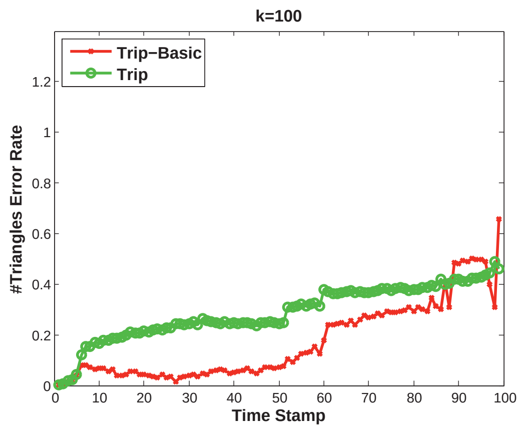


(b) k=100

Figure 2: The error rate of first eigenvector approximation.



(a) k=50

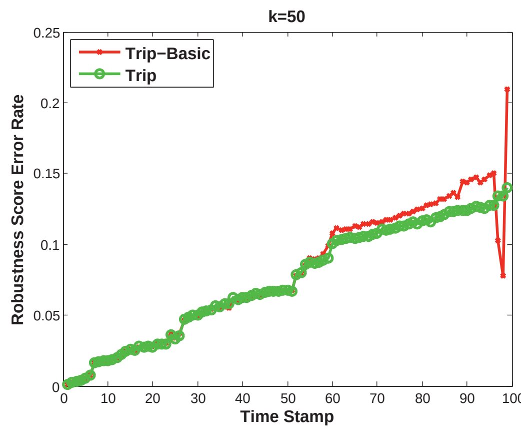


(b) k=100

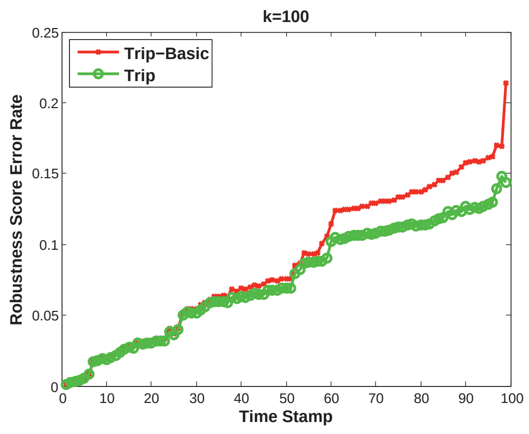
Figure 3: The error rate of number of triangles approximation.

8 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 Re-

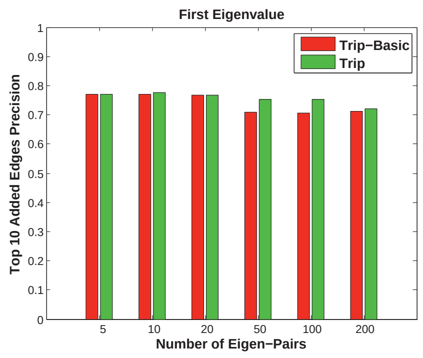


(a) k=50

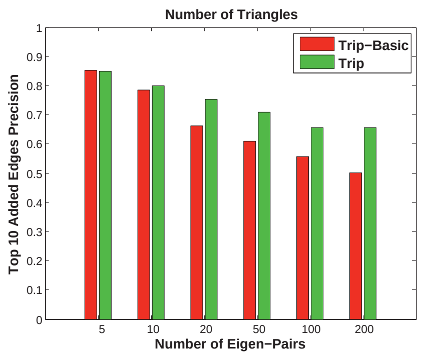


(b) k=100

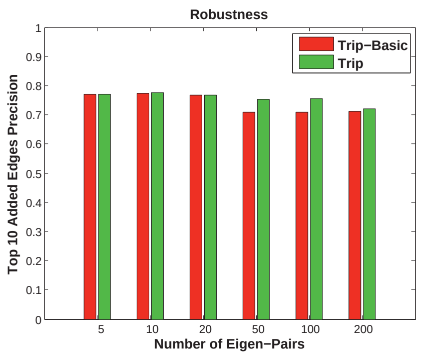
Figure 4: The error rate of robustness score approximation.



(a) First Eigenvalue

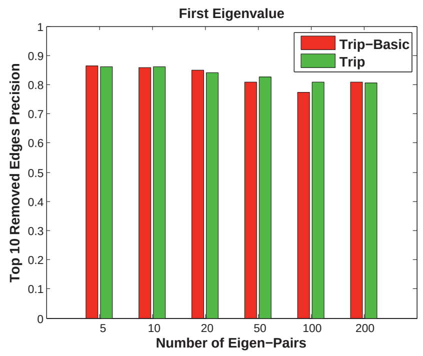


(b) Number of Triangles

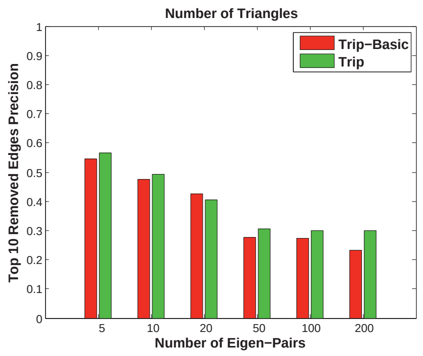


(c) Robustness

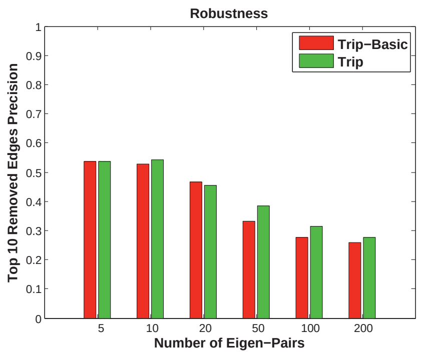
Figure 5: Average precision over time for the attribution analysis (added edges).



(a) First Eigenvalue



(b) Number of Triangles



(c) Robustness

Figure 6: Average precision over time of the attribution analysis (removed edges).

search 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] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[2] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

[3] Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *Information Theory, IEEE Transactions on*, 44(6):2743–2760, 1998.

[4] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. Epidemic thresholds in real networks. *ACM Transactions on Information and System Security (TISSEC)*, 10(4):1, 2008.

[5] Hau Chan, Leman Akoglu, and Hanghang Tong. Make it or break it: Manipulating robustness in large networks.

[6] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[7] Jure Ferlez, Christos Faloutsos, Jure Leskovec, Dunja Mladenic, and Marko Grobelnik. Monitoring network evolution using mdl. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1328–1330. IEEE, 2008.

[8] H. Frank and I. Frisch. Analysis and Design of Survivable Networks. *Communication Technology, IEEE Transactions on*, 18(5):501–519, October 1970.

[9] Ayalvadi Ganesh, Laurent Massoulié, and Don 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. Proceedings IEEE*, volume 2, pages 1455–1466. IEEE, 2005.

[10] Frank Harary and Allen Schwenk. The spectral approach to determining the number of walks in a graph. *Pacific Journal of Mathematics*, 80(2):443–449, 1979.

[11] Tsuyoshi Idé and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 440–449. ACM, 2004.

[12] Jure Leskovec, Jon Kleinberg, and Christos 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*, pages 177–187. ACM, 2005.

[13] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[14] Fragkiskos D Malliaros, Vasileios Megalooikonomou, and Christos Faloutsos. Fast robustness estimation in large social graphs: Communities and anomaly detection. In *SDM*, volume 12, pages 942–953. SIAM, 2012.

[15] Mark EJ Newman. The mathematics of networks.

[16] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[17] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[18] Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas S Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113–127, 2010.

[19] University of Oregon Route View Project. Online data and reports. `http://www.routeviews.org`.

[20] B Aditya Prakash, Deepayan Chakrabarti, Nicholas C Valler, Michalis Faloutsos, and Christos Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. *Knowledge and information systems*, 33(3):549–575, 2012.

[21] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos 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, Hyderabad, India, June 21-24, 2010. Proceedings. Part II*, pages 435–448, 2010.

[22] G. W. Stewart and Ji-Guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[23] Hanghang Tong, Spiros Papadimitriou, Philip S Yu, and Christos Faloutsos. Fast monitoring proximity and centrality on time-evolving bipartite graphs. *Statistical Analysis and Data Mining*, 1(3):142–156, 2008.

[24] Hanghang 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*, pages 245–254. ACM, 2012.

[25] Hanghang Tong, B Aditya Prakash, Charalampos Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. On the vulnerability of large graphs. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 1091–1096. IEEE, 2010.

[26] Charalampos E Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 608–617. IEEE, 2008.

[27] Charalampos E Tsourakakis. Counting triangles in real-world networks using projections. *Knowledge and Information Systems*, 26(3):501–520, 2011.

[28] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 25–34. IEEE, 2003.

[29] Stanley Wasserman. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[30] V Vassilevska Williams. Breaking the coppersmith-winograd barrier.

[31] Jun Wu, Barahona Mauricio, Yue-Jin Tan, and Hong-Zhong Deng. Natural connectivity of complex networks. *Chinese Physics Letters*, 27(7):78902, 2010.