Eigen-Optimization on Large Graphs by Edge Manipulation

CHEN CHEN and HANGHANG TONG, Arizona State University
B. ADITYA PRAKASH, Virginia Tech
TINA ELIASSI-RAD, Rutgers University
MICHALIS FALOUTSOS, University of New Mexico
CHRISTOS FALOUTSOS, Carnegie Mellon University

Large graphs are prevalent in many applications and enable a variety of information dissemination processes, e.g., meme, virus, and influence propagation. How can we optimize the underlying graph structure to affect the outcome of such dissemination processes in a desired way (e.g., stop a virus propagation, facilitate the propagation of a piece of good idea, etc)? Existing research suggests that the leading eigenvalue of the underlying graph is the key metric in determining the so-called epidemic threshold for a variety of dissemination models. In this paper, we study the problem of how to optimally place a set of edges (e.g., edge deletion and edge addition) to optimize the leading eigenvalue of the underlying graph, so that we can guide the dissemination process in a desired way. We propose effective, scalable algorithms for edge deletion and edge addition, respectively. In addition, we reveal the intrinsic relationship between edge deletion and node deletion problems. Experimental results validate the effectiveness and efficiency of the proposed algorithms.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data Mining

General Terms: Algorithm, Experimentation

Additional Key Words and Phrases: Edge manipulation, immunization, scalability, graph mining

ACM Reference Format:
Chen Chen, Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2016. Eigen-optimization on large graphs by edge manipulation. ACM Trans. Knowl. Discov. Data 10, 4, Article 49 (June 2016), 30 pages.
DOI: http://dx.doi.org/10.1145/2903148

Chen Chen and Hanghang Tong are partially supported by NSF IIS1017415, DTRA HDTRA1-16-0017, ARL W911NF-09-2-0053, NIH R01LM011986, Region II University Transportation Center under the project number 49997-33 25 and a Baidu gift. Tina Eliassi-Rad is supported from NSF CNS-1314603, DTRA HDTRA1-10-1-0120, DARPA under SMISC Program Agreement No. W911NF-12-C-0028, and under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Aditya Prakash is supported by NSF IIS-1353346, the National Endowment for the Humanities (HG-229283-15), Oak Ridge National Laboratory (Task Order 4000143330), the Maryland Procurement Office (H98230-14-C-0127), the VT College of Engineering, and a Facebook faculty gift. 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.

Authors' addresses: C. Chen, Brickyard Engineering (BYENG) 411AC, 699 S. Mill Ave. Tempe, AZ 85281; email: chen_chen@asu.edu; H. Tong, Brickyard Engineering (BYENG) 416, 699 S. Mill Ave. Tempe, AZ 85281; email: hanghang.tong@asu.edu; B. A. Prakash, 114 McBryde Hall (0106), Blacksburg VA 24061 USA; email: badityap@cs.vt.edu; T. Eliassi-Rad, 360 Huntington Ave, Mailstop 1010-177 Boston, MA 02115-5000; email: tina@eliassi.org; M. Faloutsos, Engineering Building II, Rm 332, Computer Science Department, University New Mexico, Albuquerque, New Mexico, 87131, USA; email: michalis@cs.unm.edu; C. Faloutsos, Dept. of Computer Science, GHC 8019, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891; email: christos@cs.cmu.edu.

DOI: http://dx.doi.org/10.1145/2903148

1. INTRODUCTION

Controlling the dissemination of entities on large networks is a critical problem in various disciplines, such as epidemiology, computer security, marketing, and so on. Depending on the applications, the entity could be virus, malware, or a new product to promote in the market. Various factors would affect the dissemination process, including the intrinsic properties of the entity and the topology of the network that the entity spreads on. In our work, we assume that the properties of the entity cannot be altered and focus on the topology of the graph.

For information dissemination on real graphs, a major, recent finding is that, for a family of dissemination process, the largest (in module) eigenvalue of the adjacency matrix or an appropriately defined system matrix is the *only* parameter that determines the tipping point of dissemination process (i.e., whether or not the dissemination will become an epidemic) [Prakash et al. 2012a]. Based on this finding, an optimal strategy to affect the outcome of the dissemination process should be as follows: change the graph structure (deleting/adding nodes or edges) so that the corresponding leading eigenvalue is minimized/maximized. Most previous work has studied the containment of a dissemination at the level of nodes: we can delete or inoculate nodes. At the node level, the problem of enabling a dissemination has been studied much less, since the idea of adding nodes in a network is less intuitive compared to adding edges. In more detail, there have been several works that attempt to identify which nodes we should remove or inoculate for several different propagation models, for both static [Briesemeister et al. 2003] and dynamic topologies [Prakash et al. 2010]. In addition, there have been several studies on fundamental properties of epidemic propagation for various epidemic models and topologies [Wang et al. 2003; Prakash et al. 2012a; Valler et al. 2011]. In Section 6, we review previous literature in more detail.

Despite its rationality, *node-level* manipulations might not be desirable in some applications. Consider the case of stopping malware in the network, it is not desirable to simply shut down some user accounts to contain the dissemination, since there may be some legitimate accounts among them as well. To address this issue, we shift the problem to *edge-level* manipulation, which breaks the task into *NetMelt* and *NetGel* problem. In *NetMelt* problem, we want to contain the dissemination by removing a given number of edges. For example, when a malware has been detected in the network, we may need to shut down some critical connections between routers to effectively contain its spread, which is a milder strategy than node deletion (e.g., shutting down the routers). In *NetGel* problem, on the contrary, we want to promote the dissemination by adding a given number of edges. Consider the merchants on Facebook or Twitter who want to promote their new products, they could elaborately choose a set of users to connect with and spread their promotion information via those links.

Both *NetMelt* and *NetGel* problems are challenging. For the *NetMelt* problem, most of the existing methods operate on the *node-level*, e.g., deleting a subset of the nodes from the graph to minimize the infected population from a propagating virus. In the above social spam example, this means that we have to shutdown some legitimate user accounts. Can we avoid this by operating on a finer granularity, that is, only deleting a few edges between users to slow down the social spam spreading? For the *NetGel* problem, things are even more challenging because of its high intrinsic time complexity. Let n be the number of the nodes in the graph. There are almost n^2 non-existing edges since many real graphs are very sparse. In other words, even if we only want to add one single new edge into the graph, the solution space is $O(n^2)$. This complexity "explodes" if we aim to add multiple new edges collectively, where the solution space becomes *exponential*.

The major contribution of this paper can be summarized as follows. First (*Algorithms.*), we propose effective and scalable algorithms and their variants to optimize

Table I. Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper-case)
$\mathbf{A}(i, j)$	the element at the ith row and the jth column of $\mathbf{A}$
$\mathbf{A}(i, :)$	the ith row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the jth column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{a}, \mathbf{b}, \ldots$	vectors
$\mathcal{I}, \mathcal{J}, \ldots$	sets (calligraphic)
λ	the largest (in module) eigenvalue of $\mathbf{A}$
$\mathbf{u}, \mathbf{v}$	the $n \times 1$ left eigenvector and right eigenvector associated with λ.
n	the number of the nodes in the graph
m	the number of the edges in the graph
k	the budget (i.e., the number of deleted or added edges)

the leading eigenvalue, the key graph parameter that controls the information dissemination processes for both *NetMelt* and *NetGel*, respectively. Second (*Proofs and Analysis.*), we show and prove the *hardness* of the problem; the *accuracy* and the *complexity* of our methods, and the *equivalence* between different strategies. Third (*Experimental Evaluations.*), we evaluate our approaches on real large graphs and show their effectiveness and scalability. Our evaluations on real large graphs show that our methods (a) are much more effective than the alternative choices in affecting the outcome of the dissemination processes; (b) enable a more effective way to affect the outcome of the information dissemination process by operating at the edge level; and (c) scale to large graphs.

The rest of the paper is organized as follows. We introduce notation and formal definition of *NetMelt* and *NetGel* problems in Section 2. We present and analyze the proposed algorithms in Section 3 and Section 4, respectively. The experimental evaluations are provided in Section 5. We review the related work in Section 6 and conclude the work in Section 7.

2. PROBLEM DEFINITIONS

Table I lists the main symbols used throughout the paper. We consider directed, irreducible unipartite graphs. For ease of presentation, we discuss the unweighted graph scenario although the algorithms we propose can be naturally generalized to the weighted case. We represent a graph by its adjacency matrix. Following the standard notation, we use bold upper-case for matrices (e.g., $\mathbf{A}$), bold lower-case for vectors (e.g., $\mathbf{a}$), and calligraphic fonts for sets (e.g., $\mathcal{I}$). We denote the transpose with a prime (i.e., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). Also, we represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{A}(i, j)$ is the element at the ith row and jth column of the matrix $\mathbf{A}$, and $\mathbf{A}(:, j)$ is the jth column of $\mathbf{A}$, and so on.

When we discuss the relationship between the two different strategies (node deletion vs. edge deletion) for the *NetMelt* problem, it is helpful to introduce the concept of line graph, where the nodes represent the edges in the original graph. Formally, each edge in the original graph $\mathbf{A}$ becomes a node in the line graph $L(\mathbf{A})$; and there is an edge from one node to the other in the line graph if the target of the former edge is the same as the source of the latter edge in the original graph $\mathbf{A}$. It is formally defined as follows:

Definition 1 (*Line Graph*). Given a directed graph $\mathbf{A}$, its directed line graph $L(\mathbf{A})$ is a graph such that each node of $L(\mathbf{A})$ represents an edge of $\mathbf{A}$, and there is an edge from a node e_1 to e_2 in $L(\mathbf{A})$ iff for the corresponding edges $\langle i_1, j_1\rangle$ and $\langle i_2, j_2\rangle$ are in $\mathbf{A}$, $j_1 = i_2$.

With the notation of the line graph $L(\mathbf{A})$, we have two equivalent ways to represent an edge. Let e_x ($e_x = 1, \ldots, m$) be the index of the nodes (i.e., the edges in $\mathbf{A}$) in the line graph. We can also represent the edge e_x by the pair of its source and target nodes in the original graph $\mathbf{A}$: $\langle i_x, j_x\rangle$, i.e., the edge e_x starts with the node i_x and ends at node j_x.

In order to design an effective strategy to optimize the graph structure to affect the outcome of an information dissemination process, we need to answer the following three questions: (1) *(Key graph parameters / metrics)* What are key graph metrics/parameters that determine/control the dissemination process? (2) *(Graph operations)* What types of graph operations (e.g., deleting nodes/edges, adding edges, etc) are we allowed to change the graph structure? (3) *(Affecting algorithms)* For a given graph operation, how can we design effective, scalable algorithms to optimize the corresponding key graph parameters?

For information dissemination on real graphs, a major finding [Wang et al. 2003; Prakash et al. 2012a] is that, for a large family of dissemination processes, the largest (in module) eigenvalue λ of the adjacency matrix $\mathbf{A}$ or an appropriately defined system matrix is the *only* graph parameter that determines the tipping point of the dissemination process, i.e., whether or not the dissemination will become an epidemic (see Section 6 for a review of related work). In principle, this gives a clear guidance on the algorithmic side, that is, *an ideal, optimal strategy to affect the outcome of the information dissemination process should change the graph structure so that the leading eigenvalue λ is minimized or maximized*.

Based on this observation, now we can transform the original problem of affecting the dissemination process to the eigenvalue optimization problem, that is,

(1) minimize the leading eigenvalue λ for *NetMelt*;
(2) maximize the leading eigenvalue λ for *NetGel*.

In this paper, we focus on operating on the *edge-level* to design affecting algorithms. With the above notation, our problems can be formally defined as the following two sub-problems:

PROBLEM 1. *NetMelt (Edge Deletion)*

Given. A large $n \times n$ graph $\mathbf{A}$ and an integer (budget) k;
Output. A set of k edges from $\mathbf{A}$ whose deletion from $\mathbf{A}$ creates the largest decrease of the leading eigenvalue of $\mathbf{A}$.

PROBLEM 2. *NetGel (Edge Addition)*

Given. A large $n \times n$ graph $\mathbf{A}$ and an integer (budget) k;
Find. A set of k non-existing edges of $\mathbf{A}$ whose addition to $\mathbf{A}$ creates the largest increase of the leading eigenvalue of $\mathbf{A}$.

As we will show soon, both problems are combinatorial.

3. PROPOSED ALGORITHM FOR *NETMELT*

In this section, we address the *NetMelt* problem (Prob. 1), that is, to delete k edges from the original graph $\mathbf{A}$ so that its leading eigenvalue λ will decrease as much as possible. We first study the relationship between two different strategies (edge deletion vs. node deletion), and then present our algorithm, followed by the analysis of its effectiveness as well as efficiency.

3.1. Edge Deletion vs. Node Deletion

Roughly speaking, in the *NetMelt* Problem (Edge Deletion), we want to find a set of k "important" edges from the graph $\mathbf{A}$ to delete. With the notation of the line graph $L(\mathbf{A})$, intuitively, such "important" edges in $\mathbf{A}$ might become "important" nodes in the line graph $L(\mathbf{A})$. In this section, we briefly present the relationship between these two strategies (node deletion vs. edge deletion).

Our main result is summarized in Lemma 3.1, which says that the eigenvalues of the original graph $\mathbf{A}$ are also the eigenvalues of its line graph $L(\mathbf{A})$.

LEMMA 3.1. *Line Graph Spectrum. Let λ be an eigenvalue of the graph $\mathbf{A}$. Then, λ is also the eigenvalue of the line graph $L(\mathbf{A})$.*

PROOF. See the Appendix. □

By Lemma 3.1, it seems that edge deletion (Prob. 1) can be transformed to the node deletion problem on the line graph–that is, select a subset of k nodes from the line graph $L(\mathbf{A})$ whose deletion creates the largest decrease in terms of the leading eigenvalue of $L(\mathbf{A})$. However, by the following lemma, the node deletion problem itself is still a challenging task.

LEMMA 3.2. *Hardness of Node Deletion. It is NP-Complete to find a set of k nodes from a graph $\mathbf{A}$, whose deletion will create the largest decrease of the largest eigenvalue of the graph $\mathbf{A}$.*

PROOF. The proof can be done by the reduction from the independent node set problem, which is known to be NP-Complete [Karp 1972]. See the Appendix. □

That said, we seek an effective algorithm that directly solves the *NetMelt* problem next.

3.2. Proposed K-EDGEDELETION Algorithm

The key to solving Problem 1 (*NetMelt*) is to quantify the impact of deleting a set of edges in terms of the leading eigenvalue λ. The naive way is to recompute the leading eigenvalue λ after deleting the corresponding set of edges – the smaller the new eigenvalue, the better the subset of the edges. But it is computationally infeasible for large graphs since it takes $O(m)$ time for each of the $\binom{m}{k}$ possible sets, as in general, the impact for a given set of the edges (in terms of decreasing the leading eigenvalue λ) is *not* equal to the summation of the impact of deleting each individual edge.

Let $\mathbf{u}$ and $\mathbf{v}$ be the leading left eigenvector and right eigenvector of the graph $\mathbf{A}$, respectively. Intuitively, the left eigen-score $\mathbf{u}(i)$ and the right eigen-score $\mathbf{v}(j)$ $(i, j = 1, \ldots, n)$ provide some importance measure for the corresponding nodes i and j. The core idea of the proposed K-EDGEDELETION algorithm is to quantify the impact of each edge by the corresponding left and right eigen-scores *independently* (line 9). Our upcoming analysis in the next subsection shows that this strategy (1) leads to a good approximation of the actual impact w.r.t decreasing the leading eigenvalue; and (2) naturally de-couples the dependence among the different edges. As a result, we can avoid the combinatorial enumeration in Problem 1 by picking the top-k edges with the highest individual impact scores (line 9).

According to the Perron–Frobenius Theorem [Golub and Loan 1996], the entries in the leading eigenvector share the same sign. In practice, the actual sign for the leading eigenvector could be negative for the negative leading eigenvalue. Thus, we introduce lines 2–7 in Algorithm 1 to ensure that all the eigen-scores (i.e., $\mathbf{u}(i), \mathbf{v}(j) (i, j = 1, \ldots, n)$) are non-negative. From line 8 to line 10, the score of each edge in the graph is calculated with respect to the eigen-scores of its two end nodes. Finally in line 11, the top k edges with the highest scores are returned.

ALGORITHM 1: K-EdgeDeletion

```
Input: the adjacency matrix $\mathbf{A}$ and the budget $k$
Output: $k$ edges
 1: compute the leading eigenvalue $\lambda$ of $\mathbf{A}$; let $\mathbf{u}$ and $\mathbf{v}$ be the corresponding left and right
    eigenvectors, respectively;
 2: if $\min_{i=1,\ldots,n}\mathbf{u}(i) < 0$ then
 3:     assign $\mathbf{u} \leftarrow -\mathbf{u}$
 4: end if
 5: if $\min_{i=1,\ldots,n}\mathbf{v}(i) < 0$ then
 6:     assign $\mathbf{v} \leftarrow -\mathbf{v}$
 7: end if
 8: for each edge $e_x : \langle i_x, j_x\rangle$ $e_x = 1, \ldots, m; i_x, j_x = 1, \ldots, n$ do
 9:     score$(e_x) = \mathbf{u}(i_x)\mathbf{v}(j_x)$;
10: end for
11: return top-$k$ edges with the highest score$(e_x)$
```

3.3. Proofs and Analysis

Here, we analyze the accuracy and the efficiency of the proposed K-EdgeDeletion algorithm.

The accuracy of the proposed K-EdgeDeletion is summarized in Lemma 3.3. According to Lemma 3.3, the first-order matrix perturbation theory, together with the fact that many real graphs have large eigen-gap, provides a good approximation to the impact of a set of edges in terms of decreasing the leading eigenvalue. What is more important, with such an approximation, the impact of the different edges are now de-coupled from each other. Specifically, by defining the score of each edge $e : \langle i, j\rangle$ in the original graph as $score(e) = \mathbf{u}(i)\mathbf{v}(j)$, the decrease of the leading eigenvalue caused by edge deletion can be approximated by the summation over the scores of all the deleted edges. Therefore, we can avoid the combinatorial enumeration of Problem 1 by simply returning the top-k edges with the highest individual impact scores (line 9 in Algorithm 1).

Notice that by Lemma 3.3, there is an $O(k)$ gap between the approximate and the actual impact of a set of edges in terms of decreasing the leading eigenvalue. Our experimental evaluations show that the correlation between the approximate and the actual impact is very high (See Section 5 for details), indicating that it indeed provides a good approximation for the actual decrease of the leading eigenvalue.

Lemma 3.1. *Let $\hat{\lambda}$ be the (exact) first eigenvalue of $\hat{\mathbf{A}}$, where $\hat{\mathbf{A}}$ is the perturbed version of $\mathbf{A}$ by removing all of its edges indexed by the set $\mathcal{S}$. Let $\delta = \lambda - \lambda_2$ be the eigen-gap of the matrix $\mathbf{A}$ where λ_2 is the second eigenvalue of $\mathbf{A}$, and $c = 1/(\mathbf{u}'\mathbf{v})$. If λ is the simple first eigenvalue of $\mathbf{A}$, and $\delta \geq 2\sqrt{k}$, then $\lambda - \hat{\lambda} = c\sum_{e_x\in\mathcal{S}}\mathbf{u}(i_x)\mathbf{v}(j_x) + O(k)$.*

Proof. Let $\lambda_i (i = 1, \ldots, n)$ be the ordered eigenvalues of $\mathbf{A}$ (i.e., $|\lambda| = |\lambda_1| \geq |\lambda_2| \ldots \geq |\lambda_n|$). Let $\tilde{\lambda}_i (i = 1, \ldots, n)$ be the corresponding eigenvalues of $\hat{\mathbf{A}}$. Notice that we omitted the subscripts for the leading eigenvalues (i.e., $\lambda_1 = \lambda$, and $\tilde{\lambda}_1 = \tilde{\lambda}$).

Let $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{E}$. We have $\|\mathbf{E}\|_{Fro} = \sqrt{k}$.

According to the first-order matrix perturbation theory (p. 183 [Stewart and Sun 1990]), we have

$$\begin{aligned}\tilde{\lambda}_1 &= \lambda_1 + \frac{\mathbf{u}'\mathbf{E}\mathbf{v}}{\mathbf{u}'\mathbf{v}} + \mathrm{O}(\|E\|^2)\\ &= \lambda_1 - c\sum_{e_x\in\mathcal{S}}\mathbf{u}(i_x)\mathbf{v}(j_x) + O(k). \qquad (1)\end{aligned}$$

Next, we will show that $\tilde{\lambda}_1$ is indeed the leading eigenvalue of $\hat{\mathbf{A}}$. To this end, again by the matrix perturbation theory (p. 203 [Stewart and Sun 1990]), we have

$$\begin{aligned}\tilde{\lambda}_1 &\geq \lambda_1 - \|\mathbf{E}\|_2 \geq \lambda_1 - \|\mathbf{E}\|_{Fro} \geq \lambda_1 - \sqrt{k}\\ \tilde{\lambda}_i &\leq \lambda_i + \|\mathbf{E}\|_2 \leq \lambda_i + \|\mathbf{E}\|_{Fro} \leq \lambda_i + \sqrt{k}(i \geq 2).\end{aligned} \tag{2}$$

Since $\delta = \lambda_1 - \lambda_2 \geq 2\sqrt{k}$, we have $\tilde{\lambda}_1 \geq \tilde{\lambda_i}(i = 2, \ldots, n)$. In other words, we have that $\tilde{\lambda}_1 = \hat{\lambda}$ is the leading eigenvalue of $\hat{\mathbf{A}}$. Therefore,

$$\lambda - \hat{\lambda} = c \sum_{e_x \in \mathcal{S}} \mathbf{u}(i_x)\mathbf{v}(j_x) + O(k) \tag{3}$$

which completes the proof. □

The efficiency of the proposed K-EdgeDeletion is summarized in the following lemma, which says that with a fixed budget k, K-EdgeDeletion is *linear* w.r.t the size of the graph for both time and space cost.

Lemma 3.2 (Efficiency of K-EdgeDeletion). *The time cost of Algorithm 1 is $O(mk+n)$. The space cost of Algorithm 1 is $O(n+m+k)$.*

Proof. Using the power method, line 1 takes $O(m)$ time. Lines 2–7 take $O(n)$ time. Lines 8–10 take $O(m)$ time. Line 11 takes $O(mk)$ time. Therefore, the overall time complexity of Algorithm 1 is $O(mk + n)$, which completes the proof of the time cost.

We need $O(m)$ to store the original graph $\mathbf{A}$. It takes $O(n)$ and $O(1)$ to store the eigenvectors and eigenvalue, respectively. We need additional $O(m)$ to store the scores (Line 9) for all the edges. Finally, it takes $O(k)$ for the selected k edges. Therefore, the overall space complexity of Algorithm 1 is $O(m + n + k)$, which completes the proof of the space cost. □

3.4. Variants

A. *NetMelt with Restart Strategy*

The restart strategy for *NetMelt* tends to dynamically delete the edges from the graph and then update the graph to make it ready for next batch of deletion. Given the number of edges k to be deleted from the graph, we would first break them into several batches, each of size b. The restart strategy will employ the *NetMelt* algorithm to delete b edges from current graph at each round and then update the graph until k edges are picked out. The detailed algorithm is shown in Algorithm 2.

ALGORITHM 2: K-EdgeDeletion with Restart

```
Input: the adjacency matrix A, the budget $k$ and batch size $b$
Output: $k$ edges
 1: compute the number of restarting times $t = \lfloor k/b \rfloor$
 2: initialize the edge set $\mathcal{S}$ to empty
 3: for round 1 to $t$ do
 4:    $\mathcal{S}' =$ K-EdgeDeletion($\mathbf{A}$,$b$)
 5:    $\mathcal{S} = \mathcal{S} \bigcup \mathcal{S}'$
 6:    update graph $\mathbf{A}$ by delete the edges in $\mathcal{S}'$
 7: end for
 8: if $k > tb$ then
 9:    $\mathcal{S}' =$ K-EdgeDeletion($\mathbf{A}$,$k - tb$)
10:    $\mathcal{S} = \mathcal{S} \bigcup \mathcal{S}'$
11: end if
12: return $\mathcal{S}$
```

B. *NetMelt on Weighted Graph*

On weighted graph, the matrix's leading eigenvalue change can be estimated with the same approach as in Lemma 3.3.

$$\lambda - \hat{\lambda} = c \sum_{e_x \in \mathcal{S}} \mathbf{u}(i_x)\mathbf{v}(j_x)\mathbf{w}(i_x, j_x) + O(k). \tag{4}$$

According to Equation (4), the score of each edge can be calculated as $\text{score}(e_x) = \mathbf{u}(i_x)\mathbf{v}(j_x)\mathbf{w}(i_x, j_x)$.

C. *NetMelt on Undirected Graph*

In undirected graph, the left and right eigenvectors are the same. Suppose that we only calculate the right eigenvector $\mathbf{v}$, then the leading eigenvalue change function can be modified as

$$\lambda - \hat{\lambda} = c \sum_{e_x \in \mathcal{S}} \mathbf{v}(i_x)\mathbf{v}(j_x) + O(k). \tag{5}$$

According to Equation (5), the score of each edge can be calculated as $\text{score}(e_x) = \mathbf{v}(i_x)\mathbf{v}(j_x)$.

4. PROPOSED ALGORITHM FOR *NETGEL*

In this section, we address the *NetGel* problem (Problem 2), where we want to *add* a set of new links into the graph $\mathbf{A}$ so that its leading eigenvalue λ will increase as much as possible. We first present the proposed K-EDGEADDITION algorithm, and then analyze its accuracy as well as efficiency. As most of the real world graphs are sparse, the solution space of *NetGel* problem is much larger than the one of *NetMelt* problem, which implies that solving *NetGel* would be harder than solving *NetMelt*.

4.1. Proposed K-EDGEADDITION Algorithm

Let $\mathcal{T}$ be a set of non-existing edges in $\mathbf{A}$, that is, for each $e_x : \langle i_x, j_x \rangle \in \mathcal{T}$, we have $\mathbf{A}(i_x, j_x) = 0$. Let $\hat{\lambda}$ be the leading eigenvalue of the new adjacency matrix $\hat{\mathbf{A}}$ by introducing the new edges indexed by the set $\mathcal{T}$. By the similar procedure as in the proof of Lemma 3.3, we can show that the impact of the new set of edges $\mathcal{T}$ in terms of increasing the leading eigenvalue $\hat{\lambda} - \lambda$ can be approximated as

$$\hat{\lambda} - \lambda \approx \sum_{e_x \in \mathcal{T}} \mathbf{u}(i_x)\mathbf{v}(j_x). \tag{6}$$

For any non-existing edge $e : \langle i, j \rangle$ in the original graph $\mathbf{A}$ where $\mathbf{A}(i, j) = 0$, we define its score as $score(e) = \mathbf{u}(i)\mathbf{v}(j)$. Equation (6) indicates that the increase of the leading eigenvalue caused by edge addition can be approximated by the summation over scores of all the previous non-existing added edges. Therefore, it seems that we could use a similar procedure as K-EDGEDELETION to solve the *NetGel* problem (referred to as "*Naive-Add*"): for each non-existing edge $e_x : \langle i_x, j_x \rangle$, calculate its score as $\text{score}(e_x) = \mathbf{u}(i_x)\mathbf{v}(j_x)$; and pick top-$k$ non-existing edges with the highest scores.

However, many real graphs are very sparse, i.e., $m << n^2$. Therefore, we have $O(n^2 - m) \approx O(n^2)$ possible non-existing edges. In other words, *Naive-Add* requires *quasi-quadratic* time w.r.t the number of the nodes (n) in the graph, which does not scale to large graphs.

To address this issue, we propose an efficient algorithm, which is summarized in Algorithm 3. The core idea of K-EDGEADDITION is to prune a large portion of the non-existing edge pairs based on their left and right eigen-scores. As in Algorithm 1, we take the same procedure to make sure that the left and right eigenvectors ($\mathbf{u}, \mathbf{v}$) are non-negative. We omit these steps in Algorithm 3 for brevity.

ALGORITHM 3: K-EdgeAddition

```
Input: the adjacency matrix $\mathbf{A}$ and the budget $k$
Output: $k$ non-existing edges
1: compute the left ($\mathbf{u}$) and right ($\mathbf{v}$) eigenvectors of $\mathbf{A}$ that correspond to the leading
   eigenvalue ($\mathbf{u}, \mathbf{v} \geq 0$);
2: calculate the maximum in-degree ($d_{in}$) and out-degree ($d_{out}$) of $\mathbf{A}$, respectively;
3: find the subset of $k + d_{in}$ nodes with the highest left eigen-scores $\mathbf{u}(i)$. Index them by $\mathcal{I}$;
4: find the subset of $k + d_{out}$ nodes with the highest right eigen-scores $\mathbf{v}(j)$. Index them by $\mathcal{J}$;
5: for each edge $e_x : \langle i_x, j_x \rangle \; i_x \in \mathcal{I}, j_x \in \mathcal{J}, \mathbf{A}(i_x, j_x) = 0$ do
6:     score($e_x$) $= \mathbf{u}(i_x)\mathbf{v}(j_x)$. Index them by $\mathcal{P}$;
7: end for
8: return top-$k$ non-existing edges with the highest scores among $\mathcal{P}$.
```

4.2. Proofs and Analysis

Here, we analyze the accuracy and efficiency of the proposed K-EdgeAddition.

The accuracy of the proposed K-EdgeAddition is summarized in Lemma 4.1, which says that K-EdgeAddition selects the same set of edges as *Naive-Add*.

Lemma 4.1 (Effectiveness of K-EdgeAddition). *Algorithm 3 outputs the same set of non-existing edges as Naive-Add.*

Proof. Since the maximum in-degree of graph $\mathbf{A}$ is d_{in} and the size of source node set $\mathcal{I}$ is $k + d_{in}$, then for each possible target node $j \in \mathcal{J}$, the maximum number of existing edges directing to j from $\mathcal{I}$ would be d_{in}. Therefore, there are at least k non-existing edges starting from $\mathcal{I}$ to j. Given any target node j in the graph, for any source node i', whose corresponding $\mathbf{u}(i')$ value is not among the top $k + d_{in}$ values of left eigenvector $\mathbf{u}$, we have score($\langle i', j \rangle$) $= \mathbf{u}(i')\mathbf{v}(j)$. For each node $i \in \mathcal{I}$, $\mathbf{u}(i) \geq \mathbf{u}(i')$. Therefore we have score($\langle i, j \rangle$)$\geq$score($\langle i', j \rangle$), which indicates that there are at least k non-existing edges(starting from $\mathcal{I}$ ending with $\mathcal{J}$) whose score is greater or equal to score($\langle i', j \rangle$). Cases are similar for each target node j' whose corresponding $\mathbf{v}(j')$ value is not among the top $k + d_{out}$ values of right eigenvector $\mathbf{v}$. Conclude from all above, we can see that the top k scored non-existing edges in $\mathbf{A}$ are the same with top k scored non-existing edges start from $\mathcal{I}$ and end with $\mathcal{J}$. □

The efficiency of the proposed K-EdgeAddition is summarized in the following lemma:

Lemma 4.2 (Efficiency of K-EdgeAddition). *The time cost of Algorithm 3 is $O(m + nt + kt^2)$. The space cost of Algorithm 3 is $O(n + m + t^2)$, where $t = max(k, d_{in}, d_{out})$.*

Proof. Using the power method, line 1 takes $O(m)$ time. Line 2 takes $O(m + n)$ time. Lines 3–4 take $O(n(d_{in} + k))$ and $O(n(d_{out} + k))$ time, respectively, both of which can be written as $O(nt)$. Lines 5–7 take $O((k + d_{in})(k + d_{out})) = O(t^2)$ time. Line 8 takes $O((k + d_{in})(k + d_{out})k) = O(kt^2)$. Therefore, the overall time cost is $O(m + nt + kt^2)$, which completes the proof of the time complexity.

We need $O(m)$ to store the original graph $\mathbf{A}$. It takes $O(n)$ to store the eigenvectors $\mathbf{u}$ and $\mathbf{v}$. Line 2 takes additional $O(n + 1)$ space. Lines 3–4 take $O(d_{in} + k)$ and $O(d_{out} + k)$ space, respectively, both of which can be simplified as $O(t)$. Line 5–7 take at most $O((k + d_{in})(k + d_{out})) = O(t^2)$ space. Line 9 takes $O(k)$ space. Therefore, the overall space cost (by omitting the smaller terms) is $O(m + n + kt^2)$, which completes the proof of the space complexity. □

4.3. K-EdgeAddition+

The variants in Section 3.4 can also be applied to *NetGel* problem with the same efficiency. Notice that in K-EdgeAddition, there is a quadratic term $O(t^2)$ in its time complexity function. In this section, we propose K-EdgeAddition+ to further improve the efficiency of K-EdgeAddition, especially in sparse graphs. By observing the product matrix $\mathbf{P}$ of sorted eigenvector $\mathbf{v}$ and $\mathbf{u}$, we noticed that there is at most one candidate in each row and column as maximum score edge among the rest of non-existing edges at any time. Therefore, instead of calculating all pairs of product between $\mathbf{u}(\mathcal{I})$ and $\mathbf{v}(\mathcal{J})$, and then sorting the product array and taking the top k edges as result, we examine the product scores with certain order in $\mathbf{P}$, as shown in Algorithm 5, and pick out the edge with largest product among all the current non-existing edges at each round for k times to eliminate unnecessary calculation of pair product and array sorting operations. As shown in Algorithm 4, the left and right eigenvectors of $\mathbf{A}$ are calculated in line 1. With the maximum in-degree d_{in} and out-degree d_{out} from line 2, we identify $k + d_{in}$ nodes with the highest left eigen-scores sorted in the descending order and index them by $\mathcal{I}$ in line 3. Similarly, we identify the top $k + d_{out}$ nodes with the highest right eigen-scores indexed by $\mathcal{J}$ in line 4. In lines 6–8, we first examine whether the edge with the highest score is a non-existing edge or not. If it is, we add it to the result set $\mathcal{R}$. From line 9 to 15, we gradually expand the candidate edge set based on the indices of end nodes of currently picked edge using the *CandidateExtenstion* function and store those candidate edges in max-heap $\mathcal{H}$, as shown in line 11 and line 14. In lines 12 and 13, we pick the edge with the highest score from max-heap $\mathcal{H}$ and put it into our result set $\mathcal{R}$. Last, we return the result set $\mathcal{R}$ in line 16.

ALGORITHM 4: K-EdgeAddition+

```
Input: the adjacency matrix $\mathbf{A}$ and the budget $k$
Output: $k$ non-existing edges
 1: compute the left ($\mathbf{u}$) and right ($\mathbf{v}$) eigenvectors of $\mathbf{A}$ that correspond to the leading
    eigenvalue ($\mathbf{u}, \mathbf{v} \geq 0$);
 2: calculate the maximum in-degree ($d_{in}$) and out-degree ($d_{out}$) of $\mathbf{A}$, respectively;
 3: find the subset of $k + d_{in}$ nodes with the highest left eigen-scores $\mathbf{u}(i)$. Index them by $\mathcal{I}$;
 4: find the subset of $k + d_{out}$ nodes with the highest right eigen-scores $\mathbf{v}(j)$. Index them by $\mathcal{J}$;
 5: initialize max-heap $\mathcal{H}$ and edge set $\mathcal{R}$ to empty;
 6: if $e_1 : \langle i_1, j_1 \rangle \; i_1 \in \mathcal{I}, j_1 \in \mathcal{J}, \mathbf{A}(i_1, j_1) = 0$ then
 7:     $\mathcal{R} \leftarrow \mathcal{R} + e_1$
 8: end if
 9: Initialized row and column index $i, j$ of product matrix $\mathbf{P}$ to 1, row-range in $\mathbf{P}$ to 1
10: while $|\mathcal{R}| < k$ do
11:     CandidateExtension($i, j$, row-range)
12:     $e_x : \langle i_x, j_x \rangle \leftarrow \max(\mathcal{H})$
13:     $\mathcal{R} \leftarrow \mathcal{R} + e_x$
14:     $i \leftarrow$ index of $i_x$ in $\mathcal{I}$, $j \leftarrow$ index of $j_x$ in $\mathcal{J}$
15: end while
16: return $\mathcal{R}$.
```

The efficiency of K-EdgeAddition+ is summarized in the following lemma.

Lemma 4.3 {Efficiency of K-EdgeAddition +). *For matrix with density d, the time cost of Algorithm 4 is $O(m+nt+\sqrt{d}kt \log t)$. The space cost of Algorithm 4 is $O(n+m+t)$, where $t = max(k, d_{in}, d_{out})$.*

Proof. K-EdgeAddition+ and K-EdgeAddition are the same before line 5. At line 7, the *CandidateExtension* function takes at most $O(\sqrt{d}t)$ time to find a non-existing edge at certain row/column in sparse matrix and $O(\log t)$ time to update the heap. Therefore, the overall time cost is $O(m + nt + \sqrt{d}kt \log t)$, which completes the proof of the time complexity.

ALGORITHM 5: CandidateExtension

```
Input: row and column index $i, j$ in product matrix $\mathbf{P}$, row-range
Output: add valid candidate edges to max-heap $\mathcal{H}$
 1: if $j = 1$ then
 2:    row-range ← next valid row ($i_{next}$) in the first column (i.e., index ($i_{next}$,$j$) in $\mathbf{P}$ project to a
       non-existing edge in $\mathbf{A}$)
 3:    add ($e_{next}$ : $\langle i_{next}, j\rangle$, $\mathbf{u}(i_{next})\mathbf{v}(j)$) to max-heap $\mathcal{H}$
 4: else
 5:    find next valid column ($j_{next}$) in row $i$
 6:    if $j_{next}$ < smallest column index of valid candidates above row $i$ in $\mathcal{H}$ then
 7:       add ($e_{next}$ : $\langle i, j_{next}\rangle$, $\mathbf{u}(i)\mathbf{v}(j_{next})$) to max-heap $\mathcal{H}$
 8:    end if
 9:    for each row $i'$ between $i$ and row-range do
10:       if there exist a valid candidate edge at row $i'$ in $\mathcal{H}$ then
11:          break
12:       end if
13:       find next valid column ($j'$) in row $i'$
14:       if $j'$ < smallest column index of current valid candidates above row $i'$ in $\mathcal{H}$ then
15:          add ($e_{next}$ : $\langle i', j'\rangle$, $\mathbf{u}(i')\mathbf{v}(j')$) to max-heap $\mathcal{H}$
16:       end if
17:    end for
18: end if
19: return $\mathcal{H}$.
```

In *CandidateExtension* function, the maximum number of candidate edges in max-heap $\mathcal{H}$ is t. Therefore picking k edges from max-heap $\mathcal{H}$ would only cost a space of size $O(k + t)$. Therefore, the overall space cost (by omitting the smaller terms) is $O(m + n + t)$, which completes the proof of the space complexity. □

5. EXPERIMENTAL EVALUATIONS

In this section, we provide empirical evaluations for the proposed K-EDGEDELETION, K-EDGEADDITION and K-EDGEADDITION+ algorithms. Our evaluations mainly focus on (1) the effectiveness and (2) the efficiency of the proposed algorithms.

5.1. Experimental Setup

Data sets. We used a popular set of real graphs for our experiments – the Oregon AS (Autonomous System) router graphs, which are AS-level connectivity networks inferred from Oregon route-views.[1] These were collected once a week, for nine consecutive weeks. Table II summarizes the nine graphs we used in our evaluations.

Evaluation criteria. As mentioned before, the leading eigenvalue λ of the graph is the only graph parameter that determines the epidemic threshold for a large family of information dissemination processes. Therefore, we report the change of the leading eigenvalue for the effectiveness comparison – for both *NetMelt* and *NetGel* problems. A larger change of the leading eigenvalue is better, which suggests that we can affect the outcome of the dissemination process more. In addition, we also run virus propagation simulations to compare how different methods affect the actual outcome of the propagation. For the computational cost and scalability, we report the wall-clock time.

Machine configurations. All the experiments ran on the same machine with four 2.4GHz AMD CPUs and 48GB memory, running Linux (2.6 kernel).

[1]http://topology.eecs.umich.edu/data.html.

Table II. Dataset Summary

Dataset	n	m
Oregon-A	633	2,172
Oregon-B	1,503	5,620
Oregon-C	2,504	9,446
Oregon-D	2,854	9,864
Oregon-E	3,995	15,420
Oregon-F	5,296	20,194
Oregon-G	7,352	31,330
Oregon-H	10,860	46,818
Oregon-I	13,947	61,168

Table III. Evaluations on the Approx. Quality. Larger is Better

Dataset	$k = 10$	$k = 50$	$k = 100$	$k = 500$	$k = 1000$
Oregon-A	0.999	0.997	0.995	0.973	0.924
Oregon-B	0.999	0.999	0.998	0.993	0.988
Oregon-C	1.000	0.999	0.999	0.996	0.991
Oregon-D	0.999	0.999	0.999	0.994	0.988
Oregon-E	1.000	0.999	0.999	0.998	0.995
Oregon-F	1.000	0.999	0.999	0.998	0.997
Oregon-G	1.000	0.999	0.999	0.999	0.998
Oregon-H	1.000	1.000	0.999	0.999	0.999
Oregon-I	1.000	1.000	0.999	0.999	0.999

5.2. Effectiveness of K-EdgeDeletion

Approximation Quality. For both K-EdgeDeletion and K-EdgeAddition, we want to approximate the actual change of the leading eigenvalue by the first-order matrix perturbation theory. This is the *only* place we introduce the approximation. By Lemma 3.3, it says that the quality of such an approximation depends on both the budget k as well as the eigen-gap of the original graph, with an $O(k)$ gap. Here, let us experimentally evaluate how good this approximation is on real graphs. We compute the linear correlation coefficient between the actual and approximate leading eigenvalue after we randomly remove k ($k = 10, 50, 100, 500, 1,000$) edges. The results are shown in Table III. It can be seen that the approximation is very good – in all the cases, the linear correlation coefficient is greater than 0.92, and often it is very close to 1.

The Impact of Decreasing the Leading Eigenvalue. Here, we evaluate the effectiveness of the proposed K-EdgeDeletion in terms of decreasing the leading eigenvalue λ of the graph. Lemma 3.1 suggests that the "important" edges on the original graph $\mathbf{A}$ might become "important" nodes on the line graph $L(\mathbf{A})$. We follow this intuition to design the following comparative strategies: (1) randomly select k edges from the original graph $\mathbf{A}$ (referred to as "Rand"); (2) select k edges with the highest degrees in the line graph $L(\mathbf{A})$ (referred to as "Line-Deg"); (3) select k edges with the highest eigen-scores in the line graph $L(\mathbf{A})$ (referred to as "Line-Eig"); and (4) select k edges with the highest PageRank scores in the line graph $L(\mathbf{A})$ (referred to as "Line-Page"). For "Rand", we run the experiments 100 times and report the average result. For "Line-Deg", we have two variants by using out-degree or in-degree. In our evaluation, we found that these two variants give the similar results. Therefore, we only report the results by out-degree. For the same reason, we only report the results by the right eigen-scores for "Line-Eigs". For "Line-Page", there is an additional parameter of the teleport probability. We run the experiments with the different teleport probabilities and report the best results.

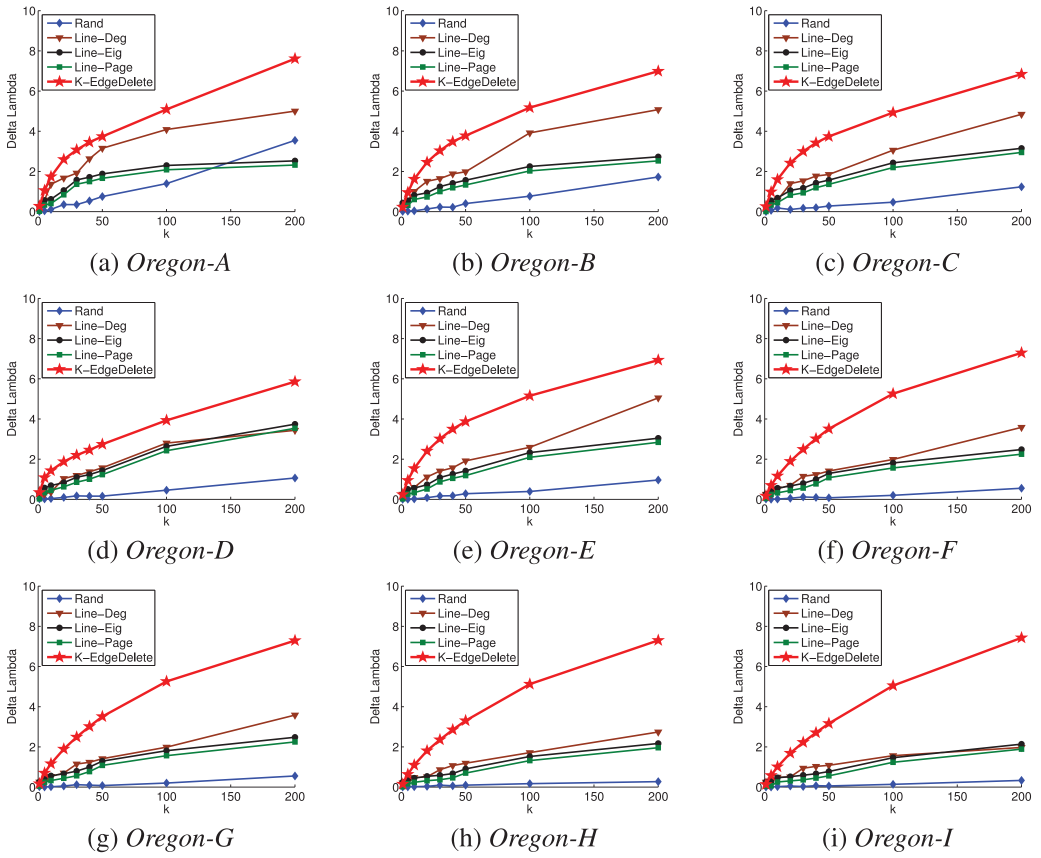


Fig. 1. The decrease of the leading eigenvalue vs. the budget k. Larger is better. The proposed K-EdgeDeletion always leads to the biggest decrease of the leading eigenvalue.

Figure 1 presents the results on all the *Oregon* graphs, it can be seen that our K-EdgeDeletion always leads to the biggest decrease in terms of the leading eigenvalue. For example, on *Oregon-C* graph, the proposed K-EdgeDeletion decreases the leading eigenvalue by *3.8* with the budget $k = 50$, which is almost double of the second best method (e.g., *2.0* by "Line-Deg"). Therefore, we expect that K-EdgeDeletion would affect the outcome of the dissemination processes better than the alternative choices, e.g., having less number of infected nodes in the graph, and so on. We validate this next.

Affecting Virus Propagation. Next, we evaluate the effectiveness of the proposed K-EdgeDeletion in terms of minimizing the outcome of the information dissemination processes. To this end, we simulate the virus propagation for the SIS model (susceptible-infective-susceptible) on the graph [Wang et al. 2003]. For each method, we delete $k = 200$ edges from the original graph. Let $s = \lambda b/d$ be the normalized virus strength (bigger s means stronger virus), where b and d are the infection rate and death rate, respectively. The results are presented in Figure 2, which is averaged over 1,000 runs. It can be seen that the proposed K-EdgeDeletion is always the best – its curve is always the lowest which means that we always, as desired, have the least number of infected nodes in the graph with this strategy. In Figure 2, "Original" (the yellow curve) means that we simulate the virus propagation on the *original* graph without deleting any

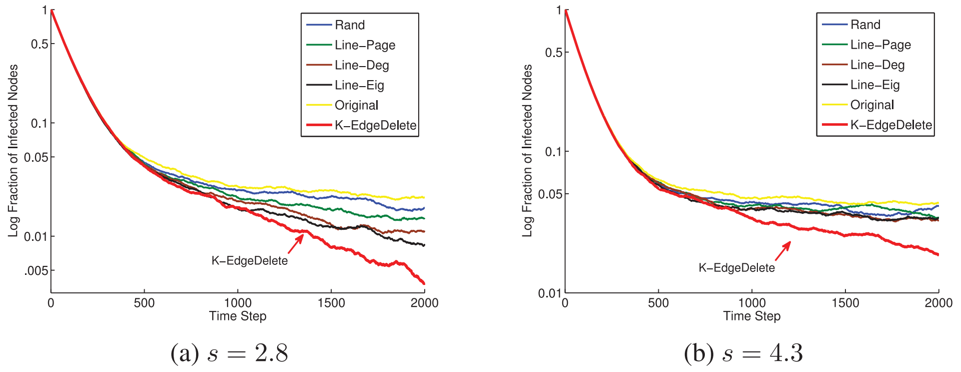


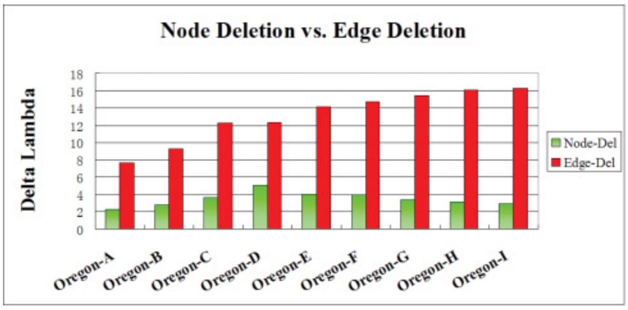
Fig. 2. Comparison of minimizing the outcome of the virus propagation. Fraction of infected nodes vs. time stamp. Lower is better. The proposed K-EdgeDeletion always leads to the least number of infected nodes. Notice that y-axis is in the logarithmic scale.

edges. Notice that when the virus becomes stronger (Figure 2(b)), all the curves except the proposed method mix with "Original", which means that they all fail to affect the virus propagation in this case. In contrast, our proposed method (the red curve) can still significantly reduce the number of infected nodes.

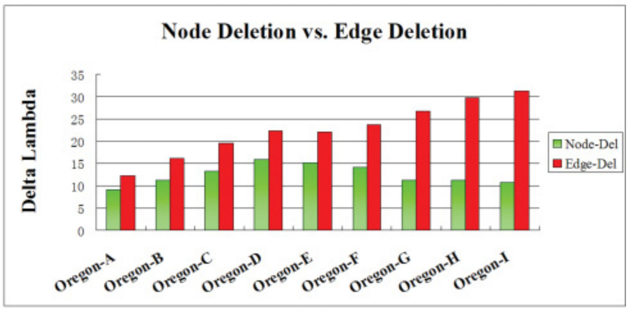
Node Deletion vs. Edge Deletion. Finally, in some applications, e.g., to stop malware propagation on the computer networks, both node deletion (e.g., shutting down some machines) and edge deletion (e.g., blocking some links between machines) are feasible. In this case, we want to know which strategy (node deletion or edge deletion) is more effective in affecting the outcome of such propagation process. To this end, we use an effective node immunization algorithm [Tong et al. 2010; Chen et al. 2016] to delete $\tilde{k} = 1, 5, 10$ nodes, respectively (referred to as "Node-Del"). For each $\tilde{k}$, we then use our proposed K-EdgeDeletion to delete the same amount of edges from the original graph (referred to as "Edge-Del"). We compare the decrease of the leading eigenvalues of the two methods. The results are summarized in Figure 3. It can be seen that "Edge-Del" always leads to a bigger decrease of the leading eigenvalue – which suggests that by operating on the edge level, we can design a more effective algorithm with the same budget to affect the outcome of the information dissemination process. The results are consistent with the intuition – not all the edges adjacent to the "important" nodes, which the node immunization algorithm aims to delete, are also "important" (e.g., many edges adjacent to an "important" node might link to/from some degree-1 nodes). In other words, edge deletion enables us to optimize the underlying graph structure on a finer granularity by picking each individual edge one by one.

5.3. Effectiveness of K-EdgeAddition

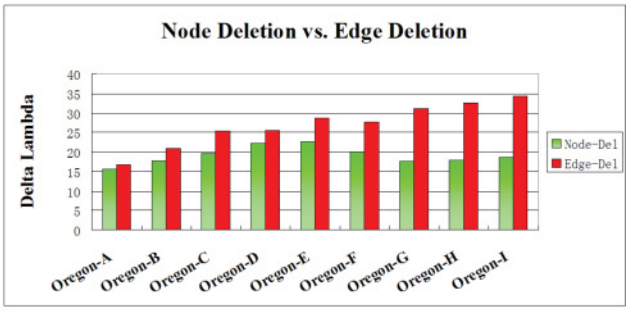
To our best knowledge, there are no existing methods to add k new links into an existing graph in order to increase its leading eigenvalue. Let $\bar{\mathbf{A}}$ be the complementary graph of $\mathbf{A}$, which has the same node set as $\mathbf{A}$, and $\bar{\mathbf{A}}(i, j) = 1$ iff $\mathbf{A}(i, j) = 0$. With the notation of the complementary graph, we use the following intuition to design the comparative methods: to select k "important" edges from the *complementary graph* $\bar{\mathbf{A}}$ and add them into the original graph $\mathbf{A}$. More specifically, we compare the proposed K-EdgeAddition with the following strategies: (1) randomly select k edges (referred to as "Rand"); (2) select k edges with the highest out-degrees in the line graph of the complementary graph $\bar{\mathbf{A}}$ (referred to as "CompDeg"); (3) select k edges with the highest right eigen-scores in the line graph of the complementary graph $\bar{\mathbf{A}}$ (referred to as



(a) $\tilde{k} = 1$



(b) $\tilde{k} = 5$



(c) $\tilde{k} = 10$

Fig. 3. Comparison between node deletion vs. edge deletion. Larger is better. With the same amount of edges deleted, our proposed K-EDGEDELETION (red) leads to a bigger decrease in terms of the leading eigenvalue.

"CompEigs"); (4) select k edges with the highest PageRank scores in the line graph of the complementary graph $\bar{\mathbf{A}}$ (referred to as "CompPage"); and (5) select k edges by running K-EDGEDELETION in the complementary graph $\bar{\mathbf{A}}$ (referred to as "CompDelete"). Again, for "Rand", we run the experiments 100 times and report the average result. We only report the results of "CompDeg" by out-degree and those of "CompEig" by right eigen-scores, respectively, since the other variants give the similar performance. For "CompPage", we run the experiments with the different teleport probabilities and report the best results.

The Impact of Increasing the Leading Eigenvalue. We first evaluate the effectiveness of the proposed K-EDGEADDITION in terms of *increasing* the leading eigenvalue of the

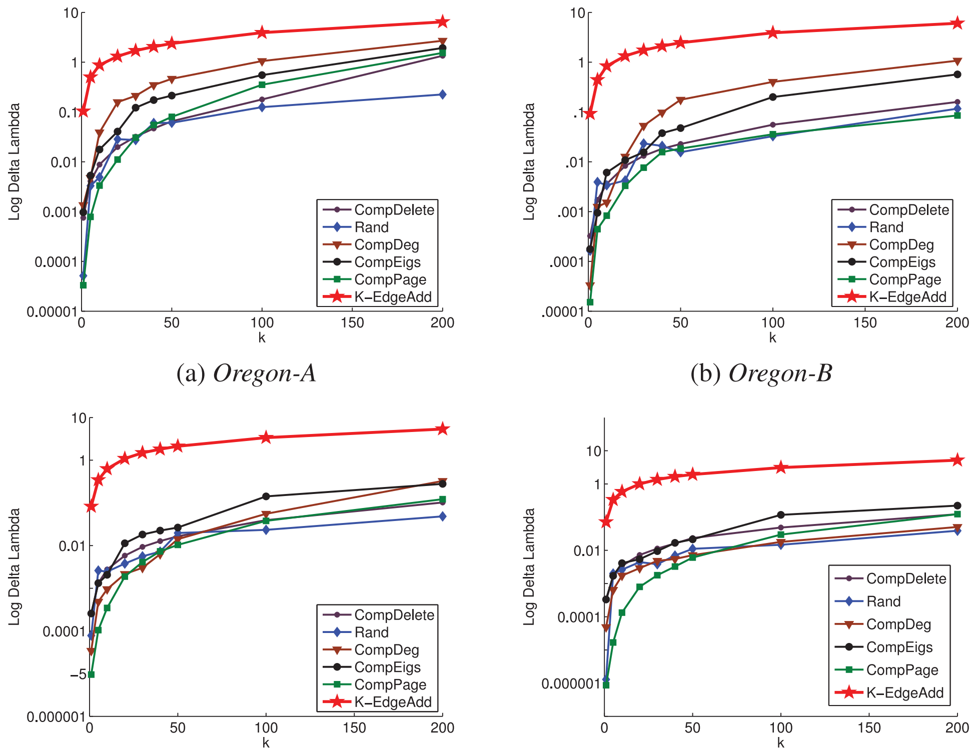
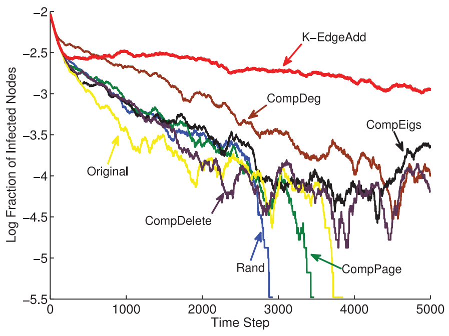


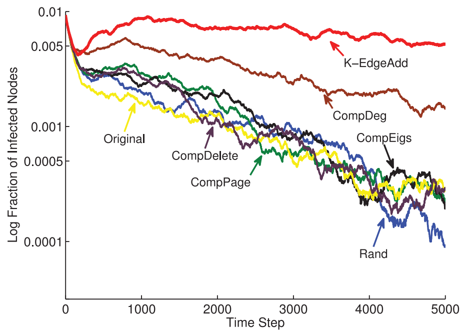
Fig. 4. The increase of the leading eigenvalue vs. the budget k. Larger is better. The proposed K-EDGEADDITION always leads to the largest increase of the leading eigenvalue. Notice that y-axis is in the logarithmic scale.

graph. From Figure 4, it can be seen that the proposed K-EDGEADDITION always leads to the biggest increase in terms of the leading eigenvalue of the graph. Notice that for all the comparative methods, they behave like "Rand" (blue curve), especially when the budget k is small.

Affecting Virus Propagation. We also evaluated the effectiveness of the proposed K-EDGEADDITION in terms of *maximizing* the outcome of the information dissemination process. To this end, again, we simulate the virus propagation for the SIS model on the graph. For each method, we add $k = 200$ new edges into the graph. Again, let $s = \lambda b/d$ be the normalized virus strength, with bigger s being stronger virus. Here, our goal is to *increase* the number of "infected" nodes (e.g., having more people in the social networks to adopt a piece of good idea, etc) by introducing a set of new links into the graph. The result is presented in Figure 5, which is averaged over 1,000 runs. It can be seen that the proposed K-EDGEADDITION is always the best – its curve is always the highest which means that we always have the largest number of "infected" nodes in the graph with this strategy. Notice that when the strength of the virus is weak (Figure 5(a)), all the curves except the proposed method mix with or are very close to "Original" (yellow curve), which means that they have little impact to boost the outcome of the propagation in this case. In contrast, our proposed method (the red curve) can still significantly increase the number of "infected" nodes. Therefore, we conclude that our proposed K-EDGEADDITION is much more effective to guide the outcome of the dissemination process.

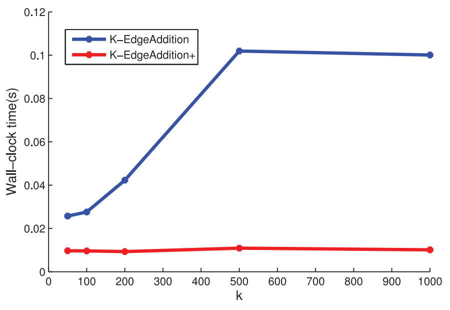


(a) $s = 1.1$

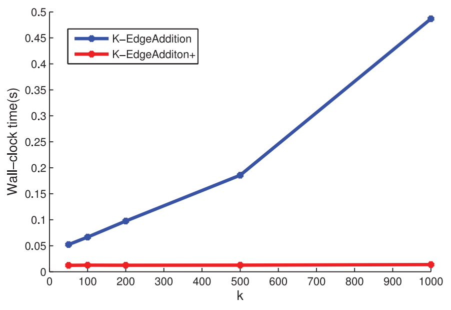


(b) $s = 1.3$

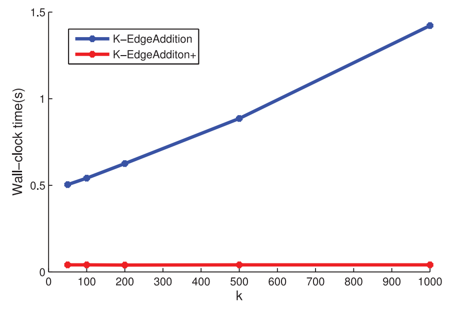
Fig. 5. Comparison of maximizing the outcome of virus propagation. Fraction of "infected" nodes vs. time stamp. Larger is better. The proposed K-EdgeAddition always leads to the largest number of "infected" nodes. Notice that y-axis is in the logarithmic scale.



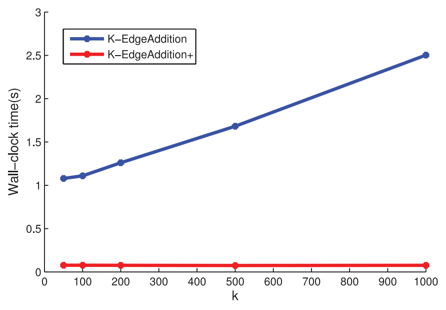
(a) *Oregon-A*



(b) *Oregon-B*



(c) *Oregon-H*

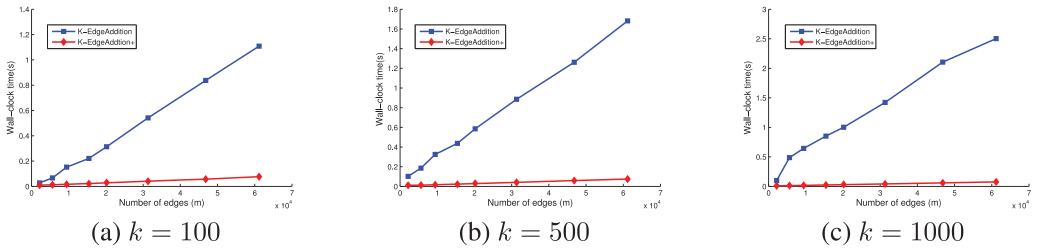


(d) *Oregon-I*

Fig. 6. The running time of K-EdgeAddition and K-EdgeAddition+ w.r.t different k.

5.4. Efficiency of K-EdgeAddition+

In this section, we compare the efficiency of K-EdgeAddition and K-EdgeAddition+. We present the experiment results on both small size graphs (*Oregon-A*,*Oregon-B*) and large size graphs (*Oregon-H*, *Oregon-I*) in Figure 6. The results on rest of the graphs in Oregon data set have similar results with the above four. From Figure 6, it is easy to see that the running time of K-EdgeAddition grows linearly w.r.t the number of edges to be added. While the K-EdgeAddition+ runs in almost constant time. It's obvious to see that K-EdgeAddition+ algorithm is much more efficient and scalable in the experiment



(a) $k = 100$ (b) $k = 500$ (c) $k = 1000$

Fig. 7. Comparison of the scalability of K-EdgeAddition and K-EdgeAddition+.

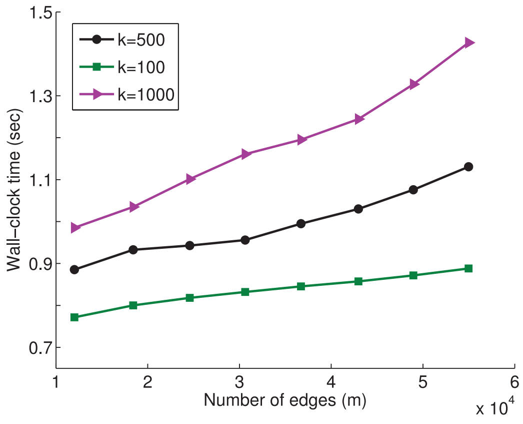


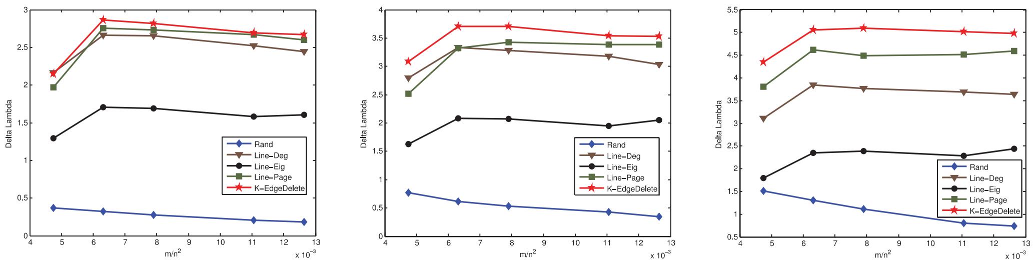
Fig. 8. Scalability of proposed algorithms. K-EdgeDeletion scales near-linearly w.r.t the size of the graph.

data set. Since the graph in *Oregon-A* only has 633 nodes, which is smaller than $1,000$, the term t in the complexity function of K-EdgeAddition is upper bounded by n. As $k = 500, 1,000$ is relatively large, it is more efficient to sort the product array first and then take the top k edges than picking out largest product pair from the array for k times. The sorting operation takes $O(tlogt)$ time no matter how large k is, which makes the size of the graph (m, n) dominate the complexity function. As a consequence, the calculation time of $k = 500$ and $k = 1,000$ for K-EdgeAddition looks almost the same on *Oregon-A*.

To compare the scalability of the two algorithms, we experimented on different sizes of graphs and the results are presented in Figure 7. The experiment is done with the same environment setting as above. Both K-EdgeAddition and K-EdgeAddition+ scale almost near-linear w.r.t m, which means they are both suitable for large graphs. We can see that the running time of K-EdgeAddition+ is almost the same when setting different k values to it. This is due to the sparsity of the graphs, which makes the size of a graph dominate the complexity function. It's obvious to see that K-EdgeAddition+ scales better than K-EdgeAddition.

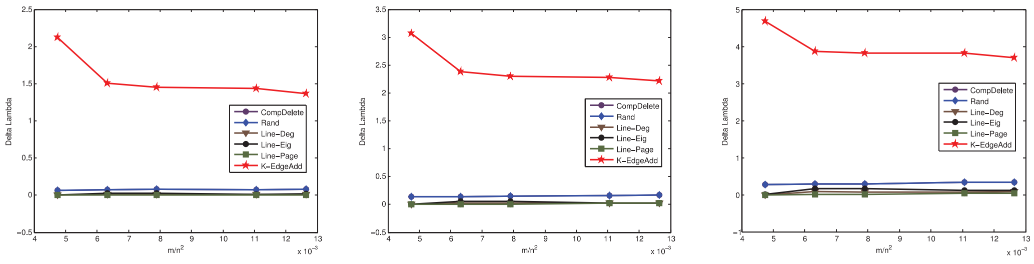
5.5. Scalability

We use the subsets of the largest data set *Oregon-I* to evaluate the scalability of the proposed algorithms. The results are presented in Figure 8. We can see that the proposed K-EdgeDeletion also scales almost near-linearly w.r.t m, which means that it is suitable for large graphs. For both K-EdgeDeletion and K-EdgeAddition cases, we also observe a slight super-linear trend. Part of the reason is due to the fact that we use the power method to compute the leading eigenvalue and the corresponding eigenvectors. When m increases, the actual iteration number in the power method also



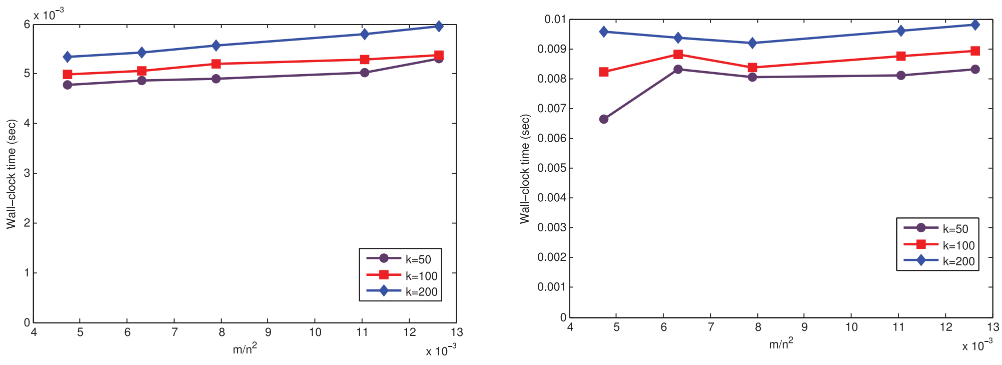
(a) $k = 50$ (b) $k = 100$ (c) $k = 200$

Fig. 9. The effectiveness of K-EDGEDELETION w.r.t. the density of the graph.



(a) $k = 50$ (b) $k = 100$ (c) $k = 200$

Fig. 10. The effectiveness of K-EDGEADDITION w.r.t. the density of the graph.



(a) K-EDGEDELETION (b) K-EDGEADDITION

Fig. 11. The efficiency of K-EDGEDELETION and K-EDGEADDITION w.r.t. the density of the graph.

tends to increase. More details about the analysis of this super-linear trend can be seen in Tong et al. [2012].

5.6. The Impact of the Graph Density

In this section, we evaluate the impact of graph density on the proposed K-EDGEDELETION and K-EDGEADDITION algorithm. The experiment results are based on *Oregon-A*, where we randomly delete and add certain number of edges to generate graphs with different densities. Figure 9 and Figure 10 show the effectiveness of K-EDGEDELETION and K-EDGEADDITION, respectively. It can be seen that the proposed methods outperform all alternative strategies consistently with different graph densities. Figure 11 shows the efficiency of K-EDGEDELETION and K-EDGEADDITION as graph density changes. It can be seen that the running time of both algorithms are stable w.r.t the graph densities, with a light linear increase for the K-EDGEDELETION algorithm.

6. RELATED WORK

In this section, we review the related work, which can be categorized into three parts: information dissemination, affecting algorithms, and node/edge importance measure.

Information Dissemination. Many research works in virus propagation have been devoted to studying the so-called epidemic threshold, that is, to determine the condition under which an epidemic will break out. While earlier works [Hethcote 2000] focus on some specific types of graph structure (e.g., random graphs, power-law graphs, etc), Wang et al. [2003] and its follow-up paper by Ganesh et al. [2005] found that, for the flu-like SIS model, the epidemic threshold for any *arbitrary, real* graph is determined by the leading eigenvalue of the adjacency matrix of the graph. Prakash et al. [2012a] further discovered that the leading eigenvalue (and a model-dependent constant) is the only parameter that determines the epidemic threshold for *all* virus propagation models (more than 25 models, including H.I.V.) in the standard literature. In this work, we aim to take one step further, i.e., how to optimize (minimize or maximize) the leading eigenvalue of the graph by deleting or adding a set of links.

There are also many research interests in studying other types of information dissemination processes on large graphs, including (a) information cascades [Bikhchandani et al. 1992; Goldenberg et al. 2001], (b) blog propagations [Leskovec et al. 2007; Gruhl et al. 2004; Kumar et al. 2005; Richardson and Domingos 2002], (c) patterns of influence propagation [Matsubara et al. 2012], (d) spreading process of competing information [Beutel et al. 2012; Valler 2012; Wei et al. 2013], and (e) viral marketing and product penetration [Kempe et al. 2003; Leskovec et al. 2006].

Affecting Algorithms. Hayashi et al. [2003] derived the extinction conditions under random and targeted immunization for the SHIR model (Susceptible, Hidden, Infectious, Recovered). Tong et al. [2010] proposed an effective node immunization strategy for the SIS model by approximately minimizing the leading eigenvalue. Briesemeister et al. [2003] studied the defending policy in power-law graphs. Prakash et al. [2010] and Valler et al. [2011] proposed effective algorithms to perform node immunization on time-varying graphs. Other algorithms to affect the outcome of the information dissemination include the influence maximization/minimization [Kempe et al. 2003; Datta et al. 2010; Chen et al. 2010], finding effectors in social networks [Lappas et al. 2010], identifying critical nodes in networks, [Wang et al. 2011; Habiba 2013; Berger-Wolf et al. 2011] and so on. With the assumption that a node can get involved in the dissemination process from more than one neighbor, Tuli et al. [2012] proposed a way to select critical nodes for both simple and complex contagions. Nguyen et al. aim to pick out critical nodes in the network under limited budget as in Nguyen and Zheng [2013] and Nguyen [2013]. More specifically, Saito et al. [2011] and Yamagishi et al. [2011] explored the attributes of nodes and applied them on building the diffusion probability model and opinion formation model, respectively. Notice that all these works focus on operating on the node level to affect the outcome of the dissemination. In contrast, we study the equally important, but much less studied affecting algorithms by operating on the edge level.

There exist some *empirical evaluations* on edge removal strategies for slightly different purposes, such as, slowing down the influenza spreading [Marcelino and Kaiser 2009], minimizing the average infection probability [Schneider et al. 2011], evaluating and comparing the attack vulnerability [Holme et al. 2002], and so on. The closest related work to our K-EDGEDELETION algorithm is Bishop and Shames [2011], which proposed a convex optimization based approach to approximately minimize the leading eigenvalue of the graph. However, the method is based on semi-definite programming and does not scale to large graphs. Moreover, for all these methods, it remains unclear if they can be generalized to address the even more challenging *NetGel* problem, where we want to *add* new edges to promote the information dissemination.

Other interesting studies include self-similar selection approach of immunization [Kim and Jung 2013], which makes it possible to do the analysis in the absence of network morphology at individual node level. The reverse engineering of immunization problem [Prakash et al. 2012b, 2013] is also a hot topic in this domain, which can be defined as: given a snapshot of a graph in which an infection has been spreading for some time, find out the original seed set where the infection started. A thorough studies of the theory about determining epidemic in the network, algorithms about effective immunization and reverse engineering are given in Prakash [2012a, 2012b].

Measuring the Importance of Nodes and Edges. In the literature, there are a lot of node importance measurements, including betweenness centrality, both the one based on the shortest path [Freeman 1977] and the one based on random walks [Newman 2005; Kang et al. 2011] PageRank [Page et al. 1998], HITS [Kleinberg 1998], and coreness score [Moody and White 2002]. Our work is also related to the so-called k-vital edges problem, which aims to delete a set of links from the graphs to increase the shortest path length [Liang et al. 2000] or the weight of the minimum spanning tree of the remaining graph [Shen 1995]. K-vital edge problem itself is known to be NP-Hard. Other remotely related work includes network inhibition [Phillips 1993] and network-interdiction [Wood 1993; Israeli and Wood 2002], both of which are also NP-Hard.

General Graph Mining. There is a lot of work on graph mining. Representative works include pattern and law mining [Broder et al. 2000], frequent substructure discovery [Xin et al. 2005], compression [Maserrat and Pei 2010], fraud and anomaly detection [Noble and Cook 2003], community mining and graph partition [Karypis and Kumar 1999; Satuluri and Parthasarathy 2009; Maiya and Berger-Wolf 2010; Yin et al. 2005], social action tracking [Tan et al. 2010], user click-through modeling [Liu et al. 2009; Agarwal et al. 2007], collaborative filtering [Koren 2009; Ge et al. 2010; Shan and Banerjee 2010; Heckerman et al. 2000], team formation [Lappas et al. 2009], network classification [Neville et al. 2009], link prediction [Liben-Nowell and Kleinberg 2007; Lichtenwalter et al. 2010; Backstrom and Leskovec 2011], sampling in graph [Maiya and Berger-Wolf 2011; Maiya 2011], within-network and across-network classification [Henderson et al. 2011], dynamic graph eigen-functions tracking [Chen and Tong 2015], multi-layered network connectivity controlling, [Chen et al. 2015] and so on.

7. CONCLUSION

In this paper, we formulate the problem of dissemination management via edge manipulation and conduct a comprehensive study on the problems. The main contributions of the paper are:

A. *Algorithms*. We take the leading eigenvalue of the graph as key parameter to control the information dissemination process. Based on this criteria, we propose effective and scalable algorithms and their variants for *NetMelt* and *NetGel*, respectively;
B. *Proofs and Analysis*. We prove the *hardness* of the problem (Lemma 3.2), the *accuracy* (Lemma 3.3 and Lemma 4.1) and the *complexity* of our methods (Lemma 3.4, Lemma 4.2, and Lemma 4.3), and the *equivalence* between different strategies (Lemma 3.1, Lemma A.1, and Lemma A.2);
C. *Experimental Evaluations*. Our evaluations on real large graphs show that (a) compared with alternative algorithms to optimize the link structure, our methods are much more effective to achieve the desired outcome of the dissemination process; (b) compared with the node deletion strategy, our K-EdgeDeletion is more effective by operating on the edge level; (c) compared with K-EdgeAddition, K-EdgeAddition+

is more efficient for real graphs; and (d) both K-EdgeDeletion and K-EdgeAddition scale to large graphs.

APPENDIX

A. HIGHER-ORDER METHOD FOR *NETMELT*

From Lemma 3.3, it can be seen that the only place we introduce the approximation in Algorithm 1 is to approximate the actual decrease of the leading eigenvalue by the first-order matrix perturbation theory. The readers might wonder if we can further improve the quality by using higher-order matrix perturbation theory, while maintaining the linear scalability of the algorithm.

We explored second-order matrix perturbation theory to approximate the actual decrease of the leading eigenvalue, and found that (1) it generates very similar results as the proposed K-EdgeDeletion algorithm and (2) it requires 5–10 times more wall-clock time. The reason might be that for the *NetMelt* problem, the first-order perturbation already gives a very good approximation. Therefore, in practice, we recommend K-EdgeDeletion for simplicity.

Nonetheless, the new algorithm based on the second-order perturbation exhibits some interesting theoretic properties. It also helps understand the relationship between edge deletion and node deletion on the algorithmic level. We present it here for the completeness.

Let $c = \frac{1}{\mathbf{u}'\mathbf{v}}$, with second-order matrix perturbation, we can approximate[2] the impact of deleting a set of edges $\mathcal{S}$ in terms of the leading eigenvalue as:

$$\lambda - \hat{\lambda} \simeq \text{Impact}(\mathcal{S}) = c\left(\sum_{e_x \in \mathcal{S}} \mathbf{u}(i_x)\mathbf{v}(j_x) - \frac{1}{2\lambda}\sum_{e_x \in \mathcal{S}, e_y \in \mathcal{S}, j_x = i_y} \mathbf{u}(i_x)\mathbf{v}(j_y)\right). \tag{7}$$

Compared with the first-order perturbation (Equation (3)), we have an additional penalized term in Equation (7): $\mathbf{u}(i_x)\mathbf{v}(j_y)$ for any two adjacent edges e_x and e_y. The intuition is to encourage the edges in the set $\mathcal{S}$ to be far away (not adjacent) from each other.

By Equation (7), the impact of different edges in the set $\mathcal{S}$ is no longer independent with each other. At the first glance, this might complicate the algorithm since now we need to optimize at the set level, that is, to find a set of edges that *collectively* maximize Equation (7). However, by the following lemma, the impact defined in Equation (7) exhibits some nice diminishing return properties.

Lemma A.1 (Second-Order Approximation Properties). *The Impact($\mathcal{S}$) defined in Equation (7) has the following properties:*

(1) *Impact(Φ) = 0, where Φ is an empty set;*
(2) *Impact($\mathcal{S}$) is monotonically non-decreasing w.r.t the set $\mathcal{S}$;*
(3) *Impact($\mathcal{S}$) is sub-modular w.r.t the set $\mathcal{S}$.*

Proof. First, when $\mathcal{S} = \Phi$, we have

$$\begin{aligned}\text{Impact}(\mathcal{S}) &= c\left(\sum_{e_x \in \Phi} \mathbf{u}(i_x)\mathbf{v}(j_x) - \frac{1}{2\lambda}\sum_{e_x \in \Phi, e_y \in \Phi, j_x = i_y} \mathbf{u}(i_x)\mathbf{v}(j_y)\right)\\ &= 0.\end{aligned} \tag{8}$$

[2]This formulas is similar as the one in Milanese et al. [2010].

In the following proof, we define $\mathcal{I}, \mathcal{J}, \mathcal{K}$ be three sets and $\mathcal{I} \subseteq \mathcal{J}$. Based on $\mathcal{I}, \mathcal{J}, \mathcal{K}$: we define $\mathcal{S} = \mathcal{I} \cup \mathcal{K}$, $\mathcal{T} = \mathcal{J} \cup \mathcal{K}$, $\mathcal{R} = \mathcal{J} \backslash \mathcal{I}$. And $\tilde{\mathbf{A}}$ is denoted as the adjacency matrix of the line graph $L(\mathbf{A})$.

By the definition of Impact($\mathcal{S}$) in (7), we have

$$\begin{aligned}
&\text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{I})\\
&\quad = \text{Impact}(\mathcal{K}) - c\left(\frac{1}{2\lambda}\sum_{e_x\in\mathcal{I},e_y\in\mathcal{K},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y) + \frac{1}{2\lambda}\sum_{e_x\in\mathcal{K},e_y\in\mathcal{I},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y)\right)\\
&\quad \geq c\left(\sum_{e_x\in\mathcal{K}}\mathbf{u}(i_x)\mathbf{v}(j_x) - \frac{1}{2\lambda}\sum_{e_x\in\mathcal{S},e_y\in\mathcal{K},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y) - \frac{1}{2\lambda}\sum_{e_x\in\mathcal{K},e_y\in\mathcal{S},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y)\right)\\
&\quad = c\left(\sum_{e_y\in\mathcal{K}}\mathbf{v}(j_y)\left(\frac{1}{2}\mathbf{u}(i_y) - \frac{1}{2\lambda}\sum_{e_x\in\mathcal{S},e_y\in\mathcal{K},j_x=i_y}\mathbf{u}(i_x)\right)\right.\\
&\qquad \left. + \sum_{e_x\in\mathcal{K}}\mathbf{u}(i_x)\left(\frac{1}{2}\mathbf{v}(j_x) - \frac{1}{2\lambda}\sum_{e_x\in\mathcal{K},e_y\in\mathcal{S},j_x=i_y}\mathbf{v}(j_y)\right)\right)\\
&\quad \geq c\left(\sum_{e_y\in\mathcal{K}}\mathbf{v}(j_y)\left(\frac{1}{2}\mathbf{u}(e_y) - \frac{1}{2\lambda}\sum_{e_x:\tilde{\mathbf{A}}(e_x,e_y)=1}\mathbf{u}(e_x)\right)\right.\\
&\qquad \left. + \sum_{e_x\in\mathcal{K}}\mathbf{u}(i_x)\left(\frac{1}{2}\mathbf{v}(e_x) - \frac{1}{2\lambda}\sum_{e_y:\tilde{\mathbf{A}}(e_x,e_y)=1}\mathbf{v}(e_y)\right)\right)\\
&\quad = c\left(\sum_{e_y\in\mathcal{K}}\mathbf{v}(j_y)\left(\frac{1}{2}\mathbf{u}(e_y) - \frac{1}{2\lambda}\lambda\mathbf{u}(e_y)\right) + \sum_{e_x\in\mathcal{K}}\mathbf{u}(i_x)\left(\frac{1}{2}\mathbf{v}(e_x) - \frac{1}{2\lambda}\lambda\mathbf{v}(e_x)\right)\right)\\
&\quad = 0.
\end{aligned} \tag{9}$$

By Equation (9), we can conclude that Impact($\mathcal{S}$) is monotonically non-decreasing w.r.t the set $\mathcal{S}$.

Next we prove that Impact($\mathcal{S}$) is sub-modular w.r.t the set $\mathcal{S}$.

$$\begin{aligned}
&\text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{I})\\
=\ &\text{Impact}(\mathcal{K}) - c\left(\frac{1}{2\lambda}\sum_{e_x\in\mathcal{I},e_y\in\mathcal{K},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y) + \frac{1}{2\lambda}\sum_{e_x\in\mathcal{K},e_y\in\mathcal{I},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y)\right)
\end{aligned} \tag{10}$$

$$\begin{aligned}
&\text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{J})\\
=\ &\text{Impact}(\mathcal{K}) - c\left(\frac{1}{2\lambda}\sum_{e_x\in\mathcal{J},e_y\in\mathcal{K},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y) + \frac{1}{2\lambda}\sum_{e_x\in\mathcal{K},e_y\in\mathcal{J},j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y)\right).
\end{aligned} \tag{11}$$

By Equations (10) and (11) we have

$$\begin{aligned}
&(\text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{I})) - (\text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{J}))\\
= \; & c\left(\frac{1}{2\lambda}\sum_{e_x\in\mathcal{R}, e_y\in\mathcal{K}, j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y) + \frac{1}{2\lambda}\sum_{e_x\in\mathcal{K}, e_y\in\mathcal{R}, j_x=i_y}\mathbf{u}(i_x)\mathbf{v}(j_y)\right)\\
\geq \; & 0.\\
\Rightarrow \; & \text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{I}) \geq \text{Impact}(\mathcal{S}) - \text{Impact}(\mathcal{J}). \qquad (12)
\end{aligned}$$

Therefore, the function Impact($\mathcal{S}$) is sub-modular w.r.t the set $\mathcal{S}$. □

Thanks to such diminishing return properties, it naturally leads to the following greedy algorithm (K-EDGEDELETION+) to find a *near-optimal* subset of edges to delete from the original graph **A**. And it can be shown that the overall time complexity of K-EDGEDELETION+ remains linear w.r.t the size of the graph.

An interesting property of Algorithm 6 is that it builds the equivalence between edge deletion and node deletion on the algorithmic level:

ALGORITHM 6: K-EDGEDELETION+

```
Input: the adjacency matrix $\mathbf{A}$ and the budget $k$
Output: $k$ edges indexed by set $\mathcal{S}$
 1: compute the first eigen-value $\lambda$ of $\mathbf{A}$; compute the corresponding left and right eigenvectors
    $\mathbf{u}$ and $\mathbf{v}$ ($\mathbf{u}, \mathbf{v} \geq 0$), respectively;
 2: initialize the set $\mathcal{S}$ to be empty;
 3: score($e_x$) = $\mathbf{u}(i_x)\mathbf{v}(j_x)$ ($e_x : \langle i_x, j_x\rangle, e_x = 1, \ldots, m$);
 4: for $k_0 = 1, \ldots, k$ do
 5:   find $e_0 = \text{argmax}_{e_x, e_x\notin\mathcal{S}}\text{score}(e_x)$;
 6:   add the new edge $e_0 : (i_0, j_0)$ into $\mathcal{S}$;
 7:   for each edge $e_y : \langle i_y, j_y\rangle$ s.t. $j_y = i_0$ do
 8:     score($e_y$) ← score($e_y$) − $1/(2\lambda)\mathbf{u}(i_y)\mathbf{v}(j_0)$;
 9:   end for
10:   for each edge $e_y : \langle i_y, j_y\rangle$ s.t. $i_y = j_0$ do
11:     score($e_y$) ← score($e_y$) − $1/(2\lambda)\mathbf{u}(i_0)\mathbf{v}(j_y)$;
12:   end for
13: end for
```

LEMMA A.2 (EQUIVALENCE OF ALGORITHM 6 TO NODE IMMUNIZATION). *Let $\mathcal{S}$ be the set of edges by running Algorithm 6 on graph* **A**; *$\mathcal{T}$ be the set of edges by running the node immunization algorithm [Tong et al. 2010] on the line graph $L(\mathbf{A})$; and $|\mathcal{S}| = |\mathcal{T}|$. We have $\mathcal{S} = \mathcal{T}$.*

PROOF. In the node immunization algorithm, the score of a node i in undirected graph (symmetric matrix **B**) at each round of computation is

$$score(i) = (2\lambda - \mathbf{B}(i,i))\mathbf{u}(i)^2 - 2\mathbf{B}(i,\mathcal{S})\mathbf{u}(\mathcal{S})\mathbf{u}(i) \qquad (13)$$

where **u** is the eigenvector of graph **B**, and $\mathcal{S}$ is the selected node set for current round. The generalized score of node i in directed graph (asymmetric matrix **B**) is

$$score(i) = (2\lambda - \mathbf{B}(i,i))\mathbf{u}(i)\mathbf{v}(i) - (\mathbf{B}(i,\mathcal{S})\mathbf{u}(i)\mathbf{v}(\mathcal{S}) + \mathbf{B}(\mathcal{S},i)\mathbf{u}(\mathcal{S})\mathbf{v}(i)). \qquad (14)$$

Let e_x be an edge of graph **A** and $\tilde{\mathbf{A}}$ be the adjacency matrix of the line graph $L(\mathbf{A})$. Based on Lemma 1 and the assumption that there is no self-reference link directed graph, The score of e_x running by node immunization algorithms on $L(\mathbf{A})$ would be

$$\begin{aligned}
&s\tilde{co}re(e_x)\\
&= (2\lambda - \tilde{\mathbf{A}}(e_x, e_x))\tilde{\mathbf{u}}(e_x)\tilde{\mathbf{v}}(e_x) - (\tilde{\mathbf{A}}(e_x, \tilde{\mathcal{T}})\tilde{\mathbf{u}}(e_x)\tilde{\mathbf{v}}(\tilde{\mathcal{T}}) + \tilde{\mathbf{A}}(\tilde{\mathcal{T}}, e_x)\tilde{\mathbf{u}}(\tilde{\mathcal{T}})\tilde{\mathbf{v}}(e_x))\\
&= (2\lambda - \tilde{\mathbf{A}}(e_x, e_x))\mathbf{u}(i_x)\mathbf{v}(j_x) - \left(\sum_{e_y \in \tilde{\mathcal{T}}, \tilde{\mathbf{A}}(e_x,e_y)=1} \mathbf{u}(i_x)\mathbf{v}(j_y) + \sum_{e_y \in \tilde{\mathcal{T}}, \tilde{\mathbf{A}}(e_y,e_x)=1} \mathbf{u}(i_y)\mathbf{v}(j_x)\right)\\
&= 2\lambda\mathbf{u}(i_x)\mathbf{v}(j_x) - \left(\sum_{e_y \in \tilde{\mathcal{T}}, \tilde{\mathbf{A}}(e_x,e_y)=1} \mathbf{u}(i_x)\mathbf{v}(j_y) + \sum_{e_y \in \tilde{\mathcal{T}}, \tilde{\mathbf{A}}(e_y,e_x)=1} \mathbf{u}(i_y)\mathbf{v}(j_x)\right)\\
&= 2\lambda\left(\mathbf{u}(i_x)\mathbf{v}(j_x) - \frac{1}{2\lambda}\left(\sum_{e_y \in \tilde{\mathcal{T}}, \tilde{\mathbf{A}}(e_x,e_y)=1} \mathbf{u}(i_x)\mathbf{v}(j_y) + \sum_{e_y \in \tilde{\mathcal{T}}, \tilde{\mathbf{A}}(e_y,e_x)=1} \mathbf{u}(i_y)\mathbf{v}(j_x)\right)\right). \qquad (15)
\end{aligned}$$

While the score of edge e_x by running Algorithm 6 at each round is

$$score(e_x) = \mathbf{u}(i_x)\mathbf{v}(j_x) - \frac{1}{2\lambda}\left(\sum_{e_y \in \tilde{\mathcal{S}}, \tilde{\mathbf{A}}(e_x,e_y)=1} \mathbf{u}(i_x)\mathbf{v}(j_y) + \sum_{e_y \in \tilde{\mathcal{S}}, \tilde{\mathbf{A}}(e_y,e_x)=1} \mathbf{u}(i_y)\mathbf{v}(j_x)\right). \qquad (16)$$

By Equations 15 and 16, we have

$$s\tilde{co}re(e_x) = 2\lambda score(e_x). \qquad (17)$$

Therefore, both algorithms would pick out the same edge e_x at each round, which in the end makes $\mathcal{S} = \mathcal{T}$ and completes the proof. □

B. PROOF FOR LEMMA 3.1

PROOF. Let $\mathbf{u}$ and $\mathbf{v}$ be the left and right eigenvectors of the graph $\mathbf{A}$ that correspond to any eigenvalue λ. We have $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ and $\mathbf{u}'\mathbf{A} = \lambda\mathbf{u}'$. Equivalently, we have:

$$\lambda\mathbf{v}(i) = \sum_{j:\mathbf{A}(i,j)=1} \mathbf{v}(j) \text{ and } \lambda\mathbf{u}(j) = \sum_{i:\mathbf{A}(i,j)=1} \mathbf{u}(i). \qquad (18)$$

Let e_x ($e_x = 1, \ldots, m$) be an edge of the graph $\mathbf{A}$. Recall that we can also represent e_x by its source and target nodes: $\langle i_x, j_x\rangle$. Let $\tilde{\mathbf{A}}$ be the adjacency matrix of the line graph $L(\mathbf{A})$. By the definition of the line graph, we have $\tilde{\mathbf{A}}(e_x, e_y) = 1$ if $j_x = i_y$, and $\tilde{\mathbf{A}}(e_x, e_y) = 0$ otherwise ($e_x, e_y = 1, \ldots, m$).

We define two $m \times 1$ vectors $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ as: $\tilde{\mathbf{u}}(e_x) = \mathbf{u}(i_x)$ and $\tilde{\mathbf{v}}(e_x) = \mathbf{v}(j_x)$ with $e_x = 1, \ldots, m$.

Next, we will show that $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ are the left and right eigenvectors of $\tilde{\mathbf{A}}$, respectively, with λ being the corresponding eigenvalue.

For any edge e_x ($e_x = 1, \ldots, m$), we have

$$\begin{aligned}
\tilde{\mathbf{A}}(e_x, :)\tilde{\mathbf{v}} &= \sum_{e_y:\tilde{\mathbf{A}}(e_x,e_y)=1} \tilde{\mathbf{v}}(\mathbf{e_y}) \\
&= \sum_{e_y:i_y=j_x} \mathbf{v}(j_y)\,(\text{all edges adjacent from}\, e_x) \\
&= \sum_{j_y:\mathbf{A}(j_x,j_y)=1} \mathbf{v}(j_y)\,(\text{all edges from node}\ j_x) \\
&= \lambda\mathbf{v}(j_y)\,(\text{due to Equation}\,(18)) \\
&= \lambda\tilde{\mathbf{v}}(e_x)\,(\text{due to definition of}\ \tilde{\mathbf{v}}).
\end{aligned} \tag{19}$$

Similarly, for any edge $e_x(e_x = 1, \ldots, m)$, we have

$$\begin{aligned}
\tilde{\mathbf{u}}'\tilde{\mathbf{A}}(:, e_x) &= \sum_{e_y:\tilde{\mathbf{A}}(e_y,e_x)=1} \tilde{\mathbf{u}}(\mathbf{e_y}) \\
&= \sum_{e_y:j_y=i_x} \mathbf{u}(i_y)\,(\text{all edges adjacent to}\, e_x) \\
&= \sum_{j_y:\mathbf{A}(j_y,i_x)=1} \mathbf{u}(i_y)\,(\text{all edges to node}\, i_x) \\
&= \lambda\mathbf{u}(i_x)\,(\text{due to Equation}\,(18)) \\
&= \lambda\tilde{\mathbf{u}}(e_x)\,(\text{due to definition of}\ \tilde{\mathbf{u}})
\end{aligned} \tag{20}$$

Putting Equation (19) and Equation (20) together, we have $\tilde{\mathbf{A}}\tilde{\mathbf{v}} = \lambda\tilde{\mathbf{v}}$ and $\tilde{\mathbf{u}}'\tilde{\mathbf{A}} = \lambda\tilde{\mathbf{u}}'$, which completes the proof. □

C. PROOF FOR LEMMA 3.2

PROOF. We consider the decision version of the K-Node deletion problem as follows.

PROBLEM 3. *K-Node Deletion (Decision Version) (DEL(G, k))*

Given. A large undirected[3] *unweighted connected graph G with n nodes and an integer k;*

Output. A subset $\mathcal{S}$ of k nodes. By deleting $\mathcal{S}$ from graph G (with adjacency matrix $\mathbf{A}$), we get a new graph $G^{(\mathcal{S})}$(with adjacency matrix $\hat{\mathbf{A}}$), in which $\lambda^{(\mathcal{S})} \leq \tau$. To make the problem easier, we proof that the problem is already NP-complete when $\tau = 0$.

First, we show that K-Node deletion problem is in NP: given subset (S) to be deleted from graph G, we can check in poly-time if the first eigenvalue of new graph $G^{(\mathcal{S})}$ is less than 0 or not.

Second, we prove that K-Node deletion problem is poly-time reducible from a known NP-complete problem, i.e., the Independent Set problem(*IND*(G, k)).

PROBLEM 4. *Independent Set problem (IND(G, k))*

Given. A large undirected unweighted connected graph $G = (V, E)$ and a number $k > 0$

Output. A set of k vertices in which no two of them are adjacent

[3]We simplify the problem on undirected graph here. Problem on undirected graph is a special case of directed graph problem and is much simpler than the latter one. The NP-Completeness of latter one implies that the former one is NP-Complete as well.

Assume the size of G is n. Given an instance of $IND(G, k)$, we create an instance $DEL(G, n-k)$ (delete $n-k$ nodes in G such that the the first eigenvalue in new graph is less or equal to 0). We now need to prove two things:

1. If there is a YES answer to $IND(G, k)$, then there is a YES answer to $DEL(G, n-k)$. The adjacency matrix of G which has YES answer to $IND(G, k)$ is

   $$\mathbf{A} = \begin{pmatrix} \mathbf{S}_{k\times k} & \mathbf{X}_{k\times(n-k)} \\ \mathbf{X}_{k\times(n-k)} & \mathbf{T}_{(n-k)\times(n-k)} \end{pmatrix}$$

   where $\mathbf{S}_{k\times k} = \mathbf{0}$, because the k nodes in S are independent to each other. By deleting the rest $n-k$ nodes in T ($T = V/S$), we have $\mathbf{X}_{k\times(n-k)} = \mathbf{0}$, $\mathbf{T}_{(n-k)\times(n-k)} = \mathbf{0}$. Therefore, the adjacency matrix for new graph $G^{(T)}$ has $\hat{\mathbf{A}} = \mathbf{0}$. Hence $\lambda^{(T)} = \lambda(\mathbf{0}) = 0$. So there is a YES answer to $DEL(G, n-k)$.
2. If there is a NO answer to $IND(G, k)$, then there is a NO answer to $DEL(G, n-k)$. Suppose we have a YES answer to $DEL(G, n-k)$. Then by deleting $n-k$ nodes from graph G (suppose they are in T), we will get new graph $G^{(T)}$ with $\lambda^{(T)} \leq 0$ where

   $$\hat{\mathbf{A}} = \begin{pmatrix} \mathbf{S}_{k\times k} & \mathbf{0}_{k\times(n-k)} \\ \mathbf{0}_{k\times(n-k)} & \mathbf{0}_{(n-k)\times(n-k)} \end{pmatrix}.$$

Since $\mathbf{S}_{k\times k} \geq \mathbf{0}$, to satisfy $\lambda^{(T)} \leq 0$, we need to have $\mathbf{S}_{k\times k} = \mathbf{0}$, which implies that all the k nodes in S are independent to each other. The conclusion is contradict with the assumption that there is a NO answer to $IND(G, k)$, therefore $DEL(G, n-k)$ can only have NO answer here.

Hence, K-node Deletion (Decision Version) is NP-complete. □

REFERENCES

Deepak Agarwal, Andrei Z. Broder, Deepayan Chakrabarti, Dejan Diklic, Vanja Josifovski, and Mayssam Sayyadian. 2007. Estimating rates of rare events at multiple resolutions. In *KDD*. San Jose, CA, USA, 16–25.

Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: Predicting and recommending links in social networks. In *WSDM*. Hong Kong, China, 635–644.

Tanya Berger-Wolf and others. 2011. Working for influence: Effect of network density and modularity on diffusion in networks. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops (ICDMW)*. IEEE, Vancouver, Canada, 933–940.

Alex Beutel, B. Aditya Prakash, Roni Rosenfeld, and Christos Faloutsos. 2012. Interacting viruses in networks: Can both survive? In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, 426–434.

Sushil Bikhchandani, David Hirshleifer, and Ivo Welch. 1992. A theory of fads, fashion, custom, and cultural change in informational cascades. *J. Pol. Economy* 100, 5 (October 1992), 992–1026.

Adrian N. Bishop and Iman Shames. 2011. Link operations for slowing the spread of disease in complex networks. *EPL (Europhysics Letters)* 95.1 (2011), 18005.

Linda Briesemeister, Patric Lincoln, and Philip Porras. 2003. Epidemic profiles and defense of scale-free networks. *WORM 2003* (Oct. 27 2003), 67–75.

Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. 2000. Graph structure in the Web. *Computer Networks* 33, 1–6 (2000), 309–320.

Chen Chen, Jingrui He, Nadya Bliss, and Hanghang Tong. 2015. On the connectivity of multi-layered networks: Models, measures and optimal control. In *ICDM*. Atlantic City, NJ, USA, 715–720.

Chen Chen and Hanghang Tong. 2015. Fast eigen-functions tracking on dynamic graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, Vancouver, BC, CA, 559–567.

Chen Chen, Hanghang Tong, B. Aditya Prakash, Charalampos E. Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. 2016. Node immunization on large graphs: Theory and algorithms. *Knowledge and Data Engineering, IEEE Transactions on* 28, 1 (2016), 113–126.

Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*. Washington DC, USA, 1029–1038.

Samik Datta, Anirban Majumder, and Nisheeth Shrivastava. 2010. Viral marketing for multiple products. In *ICDM*. Sydney, Australia, 118–127.

L. C. Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

A. Ganesh, E. L. Massouli, and D. Towsley. 2005. The effect of network topology on the spread of epidemics. In *INFOCOM*. Miami, FL, USA, 1455–1466.

Yong Ge, Hui Xiong, Alexander Tuzhilin, Keli Xiao, Marco Gruteser, and Michael J. Pazzani. 2010. An energy-efficient mobile recommender system. In *KDD*. Washington DC, USA, 899–908.

Jacob Goldenberg, Barak Libai, and Eitan Muller. 2001. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters* 12.3 (2001), 211–223.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations*. The Johns Hopkins University Press.

D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. 2004. Information diffusion through blogspace. In *WWW '04*. New York, 491–501. www.www2004.org/proceedings/docs/1p491.pdf.

Habiba Habiba. 2013. *Critical Individuals in Dynamic Population Networks*. Ph.D. Dissertation. Northwestern University.

Yukio Hayashi, Masato Minoura, and Jun Matsukubo. 2003. Recoverable prevalence in growing scale-free networks and the effective immunization. *arXiv:cond-mat/0305549 v2* (Aug. 6 2003).

David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Myers Kadie. 2000. Dependency networks for collaborative filtering and data visualization. In *UAI*. San Francisco, CA, USA, 264–273.

Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. 2011. It's who you know: Graph mining using recursive structural features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Diego, CA, USA, 663–671.

Herbert W. Hethcote. 2000. The mathematics of infectious diseases. *SIAM Rev.* 42 (2000), 599–653.

Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. 2002. Attack vulnerability of complex networks. *Physical Review E* 65.5 (2002), 056109.

Eitan Israeli and R. Kevin Wood. 2002. Shortest-path network interdiction. *Networks* 40, 2 (2002), 97–111.

U. Kang, Spiros Papadimitriou, Jimeng Sun, and Hanghang Tong. 2011. Centralities in large networks: Algorithms and observations. In *SDM*. Mesa, AZ, USA, 119–130.

Richard M. Karp. 1972. *Reducibility Among Combinatorial Problems*. Springer, US.

George Karypis and Vipin Kumar. 1999. Multilevel -way hypergraph partitioning. In *DAC*. New Orleans, LA, USA, 343–348.

D. Kempe, J. Kleinberg, and E. Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*. Washington DC, 137–146.

Byung Cho Kim and Sunghwan Jung. 2013. Effective immunization of online networks: A self-similar selection approach. *Inf. Tech. Manag.* 14, 3 (2013), 257–268.

Jon M. Kleinberg. 1998. Authoritative sources in a hyperlinked environment. In *ACM-SIAM Symposium on Discrete Algorithms*.

Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *KDD*. Paris, France, 447–456.

Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. 2005. On the bursty evolution of blogspace. In *WWW*. Chiba, Japan, 159–178.

Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *KDD*. Paris, France, 467–476.

Theodoros Lappas, Evimaria Terzi, Dimitrios Gunopulos, and Heikki Mannila. 2010. Finding effectors in social networks. In *KDD*. Washington DC, USA, 1059–1068.

Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. 2006. The dynamics of viral marketing. In *EC*. ACM Press, New York, NY, USA, 228–237.

Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie Glance, and Matthew Hurst. 2007. Cascading behavior in large blog graphs: Patterns and a model. In *Society of Applied and Industrial Mathematics: Data Mining (SDM07)*. Mineapolis, MN, USA, 551–556.

Weifa Liang, Xiaojun Shen, and Qing Hu. 2000. Finding the most vital edge for graph minimization problems on meshes and hypercubes. *International Journal of Parallel and Distributed Systems and Networks* 3.4 (2000), 197–205.

David Liben-Nowell and Jon Kleinberg. 2007. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology* 58.7 (2007), 1019–1031.

Ryan Lichtenwalter, Jake T. Lussier, and Nitesh V. Chawla. 2010. New perspectives and methods in link prediction. In *KDD*. Washington DC, 243–252.

Chao Liu, Fan Guo, and Christos Faloutsos. 2009. BBM: Bayesian browsing model from petabyte-scale data. In *KDD*. Paris, France, 537–546.

Arun S. Maiya. 2011. *Sampling and Inference in Complex Networks*. Ph.D. Dissertation. Stanford University.

Arun S. Maiya and Tanya Y. Berger-Wolf. 2010. Sampling community structure. In *WWW*. Raleigh, NC, USA, 701–710.

Arun S. Maiya and Tanya Y. Berger-Wolf. 2011. 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, San Diego, 105–113.

Jose Marcelino and Marcus Kaiser. 2009. Reducing influenza spreading over the airline network. *PLoS Currents* 1 (2009).

Hossein Maserrat and Jian Pei. 2010. Neighbor query friendly compression of social networks. In *KDD*. Washington DC, USA, 533–542.

Yasuko Matsubara, Yasushi Sakurai, B. Aditya Prakash, Lei Li, and Christos Faloutsos. 2012. Rise and fall patterns of information diffusion: Model and implications. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Beijing, 6–14.

Attilio Milanese, Jie Sun, and Takashi Nishikawa. 2010. Approximating spectral impact of structural perturbations in large networks. *Physical Review E* 81.4 (2010), 046112.

James Moody and Douglas R. White. 2002. Social cohesion and embeddedness: A hierarchical conception of social groups. *Sociological Methodology* (2002), 365–368.

Jennifer Neville, Brian Gallagher, and Tina Eliassi-Rad. 2009. Evaluating statistical tests for within-network classifiers of relational data. In *ICDM*. Miami, FL, USA, 397–406.

M. E. J. Newman. 2005. A measure of betweenness centrality based on random walks. *Soc. Networks* 27 (2005), 39–54.

Huy Nguyen. 2013. *Interactions on Complex Networks: Inference Algorithms and Applications*. Ph.D. Dissertation. University of Houston.

Huy Nguyen and Rong Zheng. 2013. On budgeted influence maximization in social networks. *IEEE J. Select. Areas Commun.* 31, 6 (2013), 1084–1094.

Caleb C. Noble and Diane J. Cook. 2003. Graph-based anomaly detection. In *KDD*. Washington DC, 631–636.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford Digital Library Technologies Project. Retrieved from http://dbpubs.stanford.edu/pub/1999-66 Paper SIDL-WP-1999-0120.

Cynthia A. Phillips. 1993. The network inhibition problem. In *STOC*. San Diego, CA, USA, 776–785.

B. Aditya Prakash. 2012a. propagation and immunization in large networks. *XRDS: Crossroads, The ACM Magazine for Students* 19, 1 (2012), 56–59.

B. Aditya Prakash. 2012b. *Understanding and Managing Propagation on Large Networks: Theory, Algorithms, and Models*. Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh.

B. Aditya Prakash, Deepayan Chakrabarti, Michalis Faloutsos, Nicholas Valler, and Christos Faloutsos. 2012a. Threshold conditions for arbitrary cascade models on arbitrary networks. *Knowledge and Information Systems* 33.3 (2012), 549–575.

B. Aditya Prakash, Hanghang Tong, Nicholas Valler, Michalis Faloutsos, and Christos Faloutsos. 2010. Virus propagation on time-varying networks: Theory and immunization algorithms. In *ECML/PKDD (3)*. Barcelona, Spain, 99–114.

B. Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. 2012b. Spotting culprits in epidemics: How many and which ones?. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, Brussels, Belgium, 11–20.

B. Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. 2013. Efficiently spotting the starting points of an epidemic in a large graph. *Knowl. Info. Sys.* (2013), 1–25.

M. Richardson and P. Domingos. 2002. Mining knowledge-sharing sites for viral marketing. *SIGKDD*. Edmonton, AB, CA, 61–70.

Kazumi Saito, Kouzou Ohara, Yuki Yamagishi, Masahiro Kimura, and Hiroshi Motoda. 2011. Learning diffusion probability based on node attributes in social networks. In *Foundations of Intelligent Systems*. Springer, 153–162.

Venu Satuluri and Srinivasan Parthasarathy. 2009. Scalable graph clustering using stochastic flows: Applications to community discovery. In *KDD*. Paris, France, 737–746.

Christian M. Schneider, Tamara Mihaljev, Shlomo Havlin, and Hans J. Herrmann. 2011. Restraining epidemics by improving immunization strategies. *CoRR* abs/1102.1929 (2011).

Hanhuai Shan and Arindam Banerjee. 2010. Generalized probabilistic matrix factorizations for collaborative filtering. In *ICDM*. Sydney, Australia, 1025–1030.

Hong Shen. 1995. Finding the k most vital edges with respect to minimum spanning tree. *Acta Inform.* 36 (1995), 405–424.

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

Chenhao Tan, Jie Tang, Jimeng Sun, Quan Lin, and Fengjiao Wang. 2010. Social action tracking via noise tolerant time-varying factor graphs. In *KDD*. Washington DC, 1049–1058.

Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2012. Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. ACM, Maui, Hawaii, 245–254.

Hanghang Tong, B. Aditya Prakash, Charalampos E. Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. 2010. On the vulnerability of large graphs. In *ICDM*. Sydney, Australia, 1091–1096.

Gaurav Tuli, Chris J. Kuhlman, Madhav V. Marathe, S. S. Ravi, and Daniel J. Rosenkrantz. 2012. Blocking complex contagions using community structure. In *Workshop Multiagent Interaction Netw*.

Nicholas Valler, B. Aditya Prakash, Hanghang Tong, Michalis Faloutsos, and Christos Faloutsos. 2011. Epidemic spread in mobile ad hoc networks: Determining the tipping point. In *Networking (1)*. 266–280.

Nicholas Charles Valler. 2012. *Spreading Processes on Networks Theory and Applications*. Ph.D. Dissertation. UNIVERSITY OF CALIFORNIA.

Dashun Wang, Zhen Wen, Hanghang Tong, Ching-Yung Lin, Chaoming Song, and Albert-László Barabási. 2011. Information spreading in context. In *Proceedings of the 20th International Conference on World Wide Web*. ACM, Hyderabad, India, 735–744.

Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. 2003. Epidemic spreading in real networks: An eigenvalue viewpoint. In *SRDS*. 25–34.

Xuetao Wei, Nicholas C. Valler, B. Aditya Prakash, Iulian Neamtiu, Michalis Faloutsos, and Christos Faloutsos. 2013. Competing memes propagation on networks: A network science perspective. *Selected Areas in Communications, IEEE Journal on* 31, 6 (2013), 1049–1060.

R. Kevin Wood. 1993. Network interdiction problem. *Math. Com. Model.* 17, 2 (1993), 1–18.

D. Xin, J. Han, X. Yan, and H. Cheng. 2005. Mining compressed frequent-pattern sets. In *VLDB*. Trondheim, Norway, 709–720.

Yuki Yamagishi, Kazumi Saito, Kouzou Ohara, Masahiro Kimura, and Hiroshi Motoda. 2011. Learning attribute-weighted voter model over social networks. *J. Mach. Learn. Res.-Proc. Track* 20 (2011), 263–280.

Xiaoxin Yin, Jiawei Han, and Philip S. Yu. 2005. Cross-relational clustering with user's guidance. In *KDD*. Chicago, IL, USA, 344–353.

Received April 2014; revised April 2015; accepted March 2016