MET: A Fast Algorithm for Minimizing Propagation in Large Graphs with Small Eigen-Gaps*

Long T. Le
Rutgers University
longtle@cs.rutgers.edu

Tina Eliassi-Rad
Rutgers University
eliassi@cs.rutgers.edu

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

Abstract

Given the topology of a graph G and a budget k, how can we quickly find the best k edges to delete that minimize dissemination in G? Stopping dissemination in a graph is important in a variety of fields from epidemiology to cyber security. The spread of an entity (e.g., a virus) on an arbitrary graph depends on two properties: (1) the topology of the graph and (2) the characteristics of the entity. In many settings, we cannot manipulate the latter, such as the entity's strength. That leaves us with modifying the former (e.g., by removing nodes and/or edges from the graph in order to reduce the graph's connectivity). In this work, we address the problem of removing edges. We know that the largest eigenvalue of the graph's adjacency matrix is a good indicator for its connectivity (a.k.a. path capacity). Thus, algorithms that are able to quickly reduce the largest eigenvalue of a graph often minimize dissemination on that graph. However, a problem arises when the differences between the largest eigenvalues of a graph are small. This problem, known as the *small eigen-gap problem*, occurs often in social graphs such as Facebook postings or instant messaging (IM) networks. We introduce a scalable algorithm called *MET* (short for *Multiple Eigenvalues Tracking*), which efficiently and effectively solves the small eigen-gap problem. Our extensive experiments on different graphs from various domains show the efficacy and efficiency of our approach.

1 Introduction

In recent years, algorithms that minimize the spread of entities (such as viruses) on graphs have attracted significant attention [7, 13, 10, 12]. From the literature, we know that the *tipping point* for an entity's spread is a function of (1) the connectivity of the graph and (2) the strength of the entity. The former is often approximated by the largest eigenvalue of the adjacency matrix. The latter is exogenous to the network and is frequently measured by the birth and death rates of the entity. In this work, we assume that we cannot manipulate the strength of the entity. Thus, to minimize dissemination on a given graph, we focus on selecting the best k edges to delete. Removing edges (e.g., un-friending two users) is more palatable than removing nodes (e.g., deleting a user's account). The formal definition of our problem is as follows: Given a budget k and a graph G represented by its adjacency matrix $\mathbf{A}$, identify the k edges in $\mathbf{A}$ whose deletions will create the largest drop in the leading eigenvalue of $\mathbf{A}$.[1] This problem is NP-hard [12].

In our work, we observed that existing algorithms perform poorly on graphs with small *eigen-gaps*.[2] The reason for this poor performance is because existing methods only track the largest eigenvalue, while removing an edge affects *all* eigenvalues. In cases where the eigen-gap is small (which covers most social graphs, see Figure 2), the largest eigenvalue drops quickly while the second largest eigenvalue drops slowly. This may lead to their values being inverted as more edges are removed. We address this shortcoming by introducing *MET* (short for *Multiple Eigenvalues Tracking*). *MET* is a scalable algorithm that tracks multiple eigenvalues while deleting edges. *MET* automatically determines how many eigenvalues should be tracked and has linear runtime in the number of edges of the graph.

In our extensive experiments, we compare *MET* with seven competing methods on various social and technological graphs. We observe that *MET* is able to strike the best balance between *efficacy* (as measured by the percentage drop in the leading eigenvalue) and *efficiency* (as measured by runtime). In particular, when compared with *NetMelt*$^+$, the strongest competing method that does eigen-computation after each edge deletion, *MET* is at least $12\times$ faster while preserving at least 96% of *NetMelt*$^+$'s efficacy.

Our **contributions** are as follows: **(1)** Existing approaches that approximate the decrease in the leading eigenvalue after k edge deletions perform poorly in graphs with small eigen-gaps. This covers most social graphs. **(2)** We introduce a novel algorithm, called *MET*, which tracks mul-

*This work was funded in part by LLNL under Contract DE-AC52-07NA27344, by NSF CNS-1314603, by DTRA HDTRA1-10-1-0120, and by DAPRA under SMISC Program Agreement No. W911NF-12-C-0028.

[1] Both the 'largest eigenvalue' and the 'leading eigenvalue' refer to the biggest eigenvalue in magnitude.

[2] Eigen-gap is the difference between two consecutive eigenvalues (e.g., the difference between the largest and the second largest eigenvalues).

tiple eigenvalues. *MET* automatically chooses how many eigenvalues to track; it is scalable; and it performs well on different types of graphs with a wide range of eigen-gaps: from small to large. **(3)** Experiments on various real graphs highlight the efficacy and efficiency of our method *MET*.

The rest of the paper is organized as follows. We cover some background information in Section 2. Sections 3 and 4 describe our proposed method and our experiments, respectively. Section 5 presents the related work. We conclude the paper in Section 6.

2 Background

Table 1 lists the notations used in this paper. We represent a graph by its adjacency matrix, which is in bold upper-case letter. We use $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq$ ... to represent the largest (in magnitude) eigenvalues of an adjacency matrix. The left and right eigenvectors of the adjacency matrix are denoted by $\mathbf{u}_i$ and $\mathbf{v}_i$, respectively.

Symbol	Definition
A, **B**, ...	matrices (capital letters in boldface)
$\mathbf{A}(i,j)$	the $(i,j)^{\text{th}}$ element of matrix **A**
$\mathbf{A}(i,:)$	the i^{th} row of matrix **A**
$\mathbf{A}(:,j)$	the j^{th} column of matrix **A**
n	the number of nodes in graph
m	the number of edges in graph
k	the budget (in terms of the number of edges that we can delete)
$\lambda_1, \lambda_2, \lambda_3, ...$	eigenvalues of adjacency matrix: $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq ..$
$\mathbf{u}_i, \mathbf{v}_i$	the left and right eigenvector corresponding to λ_i

Table 1: Notations used in the paper.

Chakrabarti *et al*. [3] prove that an entity will "die out" on an undirected graph if the strength of that entity (as measured by the ratio of its birth rate β over its death rate δ) is less than one over the largest eigenvalue λ_1 of the graph's adjacency matrix−i.e., if $\frac{\beta}{\delta} < \frac{1}{\lambda_1}$. Subsequent literature [10] generalizes this result to 25 different virus models; and found that for all of these models, the larger the λ_1, the smaller the epidemic threshold. Thus, minimizing λ_1 is crucial to minimizing dissemination in graphs.

Our proposed method, *MET*, utilizes *eigenscores* to estimate the effects of removing edges on eigenvalues [12]. Given an eigenvalue and its corresponding left and right eigenvectors, the eigenscore of an edge e from node i to node j is the product of the i^{th} and j^{th} elements of the left and right eigenvectors. That is, $getEigenscore(e : i \rightarrow j, \lambda, \mathbf{u}, \mathbf{v}) = \mathbf{u}(i) \times \mathbf{v}(j)$.

Using the Power Iteration Method [12], we compute the eigenscores for the leading eigenvalue in $O(n+m)$, where n and m are the number of nodes and edges in the graph, respectively. To compute the top-T eigenscores (corresponding to the top-T eigenvalues), we use the Iterative Approximate Method by Lanczos [9], which takes $O(mT + nT^2)$.

3 Proposed Method: *MET*

All eigenvalues are affected when an edge is removed from a graph. When the graph has small eigen-gaps,[3] algorithms that *only* estimate the leading eigenvalue can exhibit significant performance degradations. One can alleviate these shortcomings in two ways: (1) track/estimate more than one eigenvalue, and (2) re-compute eigenscores of the remaining edges periodically. In the former, the question becomes how many eigenvalues should one track. In the latter, the question becomes how often should one re-compute eigenscores. Bad answers to these questions can adversely affect the efficiency and efficacy of one's algorithm. *MET*, described in Algorithm 1, provides satisfactory answers to both of these questions.

Tracking more than one eigenvalue. *MET adaptively* determines the number of eigenvalues T to track. It starts by tracking the top-2 eigenvalues (i.e., $T = 2$) and modifies T only when the gap between λ_1 and λ_T changes. If the gap decreases, then *MET* increases T by 1; if the gap increases, then *MET* decreases T by 1; otherwise, T does not change. In *MET*, the minimum value for T is 2. Note that a large T value can cause unnecessary tracking of too many eigenvalues and lead to increased runtime.

Recomputing eigenscores periodically. *MET* uses λ_T as a threshold for deciding when to re-compute eigenscores. Specifically, when all estimated λ values (i.e., $\lambda_1, \ldots, \lambda_{T-1}$) fall below the threshold λ_T, then *MET* re-computes the eigenscores. Since a small T value can cause unnecessary re-computation of eigenscores (leading to a longer runtime), *MET* uses a slack variable ϵ, whereby *MET* re-computes the eigenscores only if all estimated λ values are less than $\lambda_T - \epsilon$. A nonzero ϵ is useful when all the top T eigenvalues are very close to each other. We evaluate the effects of various ϵ values in Section 4.

LEMMA 3.1. *The runtime complexity for MET is* $O(k_1\bar{T} \times (m + n\bar{T}) + k \times (m + \bar{T}))$. *The space cost for MET is* $O(m \times (1+\bar{T}))$. *n and m are the number of nodes and edges in the graph, respectively; k is the budget for edge deletions; k_1 is the number of times that MET re-computes eigenvalues ($k_1 < k$); and $\bar{T}$ is the average number of eigenvalues that MET tracks.*

Proof. In Algorithm 1, the following lines do **not** take $O(1)$. The $while$ loop on Line 4 is executed on average k_1 times, which is also the number of times that *MET* re-computes

[3] Figure 2 shows that the eigen-gap between the top-21 eigenvalues can be as small as 0.05 in a social graph (such as Facebook user-postings).

Algorithm 1 MET Algorithm

Input:

- A graph represented by its adjacency matrix **A**
- Budget k

Output: The list of k edges to be deleted from **A**

1: $E \leftarrow \{\}$
2: $T \leftarrow 2$
3: $previousGap \leftarrow \infty$
4: **while** ($|E| < k$) **do**
5: Compute the top-T eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_T$; and the corresponding left $\mathbf{u}_i$ and right $\mathbf{v}_i$ eigenvectors for $i = 1, 2, \ldots, T-1$.
6: $Eigenscores[e, i] \leftarrow getEigenscore(e, \lambda_i, \mathbf{u}_i, \mathbf{v}_i)$, for each edge e and for $i = 1, 2, \ldots, T-1$; see Section 2.
7: $currentGap \leftarrow \lambda_1 - \lambda_T$
8: $threshold \leftarrow \lambda_T - \epsilon$
9: $\lambda_{max} \leftarrow max\{\lambda_1, \lambda_2, \ldots, \lambda_{T-1}\}$
10: **while** ($\lambda_{max} > threshold$ **and** $|E| < k$) **do**
11: $e_{max} \leftarrow$ edge with the maximum eigenscore under λ_{max} and its corresponding left and right eigenvectors.
12: $E \leftarrow E \cup e_{max}$; and remove e_{max} from **A**.
13: **for** $i = 1, 2, \ldots, T-1$ **do**
14: $\lambda_i \leftarrow \lambda_i - Eigenscores[e_{max}, i]$ // Update λ_i
15: $Eigenscores[e_{max}, i] \leftarrow -1$ // Mark e_{max} as deleted
16: **end for**
17: $\lambda_{max} \leftarrow max\{\lambda_1, \lambda_2, \ldots, \lambda_{T-1}\}$
18: **end while**
19: **if** ($currentGap < previousGap$) **then**
20: $T \leftarrow T + 1$
21: **else if** ($currentGap > previousGap$ **and** $T > 2$) **then**
22: $T \leftarrow T - 1$
23: **else**
24: // T does not change.
25: **end if**
26: $previousGap \leftarrow currentGap$
27: **end while**
28: **return** E

eigenvalues and eigenscores. Line 5 computes the average $\bar{T}$ eigenvalues and eigenvectors using Lanczos algorithm [9]; this takes $O(m\bar{T} + n\bar{T}^2)$. Line 6 computes eigenscores, which takes $O(m\bar{T})$. Line 9 takes $O(\bar{T})$ to find the maximum λ in the estimated eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_{\bar{T}}\}$. Let k_2 denote the average number of iterations of the $while$ loop on Line 10. (Note that the edge-deletion budget k is equal to $k_1 \times k_2$.) Line 11 takes $O(m)$ to find the edge with the highest eigenscore. The for loop on Lines 13 through 16 takes $O(\bar{T})$. Similar to Line 9, Line 17 takes $O(\bar{T})$ to find the maximum λ. Thus, the $while$ loop covering Lines 10 through 18 takes $O(k_2 \times (m + \bar{T} + \bar{T})) \approx O(k_2 \times (m + \bar{T}))$. Putting all of this together, we get that the total runtime of *MET* is:

$$O(k_1 \times (m\bar{T} + n\bar{T}^2 + m\bar{T} + \bar{T} + k_2 \times (m + \bar{T}))) \approx$$
$$O(k_1 m\bar{T} + k_1 n\bar{T}^2 + k_1 m\bar{T} + k_1\bar{T} + k \times (m + \bar{T})) \approx$$
$$O(k_1\bar{T} \times (m + n\bar{T} + m + 1) + k \times (m + \bar{T})) \approx$$
$$O(k_1\bar{T} \times (m + n\bar{T}) + k \times (m + \bar{T}))$$

MET needs $O(m)$ to store the graph and $O(k)$ to store the deleted edges. On average, *MET* tracks $\bar{T}$ eigenvalues and their corresponding eigenscores, this requires $O(m\bar{T})$ space. Thus, *MET* requires $O(m + k + m\bar{T})$ storage. Since $k << n$, the total storage cost for *MET* is $O(m \times (1 + \bar{T}))$.

In the runtime complexity of *MET*, there are terms $\bar{T}^2$ and k_1. Table 3 shows that $\bar{T}$ and k_1 are very small. In addition, $k << m$. Thus, the runtime of *MET* is linear with the size of graph.

MET-Naive. If one happens to know how many eigenvalues T to track *a priori*, then *MET* does not need to adaptively track multiple eigenvalues. We have a naive version of *MET*, called *MET-naive*, where k edges are sequentially deleted based on the largest estimated λ_i in $i = 1, 2, \ldots, T$. Similar to *MET*, all estimated λ values are updated after each edge deletion (Line 14 in Algorithm 1); and the eigenscore of the deleted edge is marked, so that it is not selected again (Line 15 in Algorithm 1).

MET-Naive tracks a constant number of eigenvalues T and does not re-compute eigenscores. Its runtime cost is $O(T \times (m + nT) + k \times (m + T))$. For brevity, we have omitted the proof for it here. Similar to *MET*, the space cost for *MET-Naive* is $O(m \times (1 + T))$.

See Section 4.4 for a discussion of how far *MET*'s results are from the upper-bound on the optimal solution that minimizes the largest eigenvalue of a graph.

4 Experiments

This section is organized as follows: data description, experimental setup, results, and discussion.

4.1 Data Description Table 2 lists the graphs used in our experiments. All of our graphs are undirected and unweighted. We use four different graph types to evaluate our

Dataset	Time Span	# of Nodes (n)	# of Edges (m)	Avg. Degree	# of Connected Components	% of Nodes in LCC	Average Distance in LCC
Oregon-1	7 days	5,296	10,097	3.81	1	100%	3.5
Oregon-2	7 days	7,352	15,665	4.26	1	100%	3.5
Oregon-3	7 days	10,860	23,409	4.31	1	100%	3.6
Oregon-4	7 days	13,947	30,584	4.39	1	100%	3.6
YIM-1	1 day	50,492	79,219	3.14	2802	54.5%	16.2
YIM-2	1 day	56,454	89,269	3.16	2650	61.2%	15.9
YIM-3	1 day	31,420	39,072	2.49	2820	62.9%	13
FB-1	1 day	27,165	26,231	1.93	2081	50.4%	9.4
FB-2	1 day	29,556	29,497	1.99	1786	57.8%	9.3
FB-3	1 day	29,702	29,384	1.98	1902	55.6%	9.4
FB-4	1 day	29,442	29,505	2.00	1746	58.1%	8.9
FB-5	1 day	29,553	29,892	2.02	1624	59.8%	8.8
Twitter-1	1 month	25,799	16,410	2.18	2899	63.6%	7.5
Twitter-2	1 month	39,477	45,149	2.29	3827	68.9%	7.3
Twitter-3	1 month	57,235	68,010	2.38	4828	73.4%	7.5
Twitter-4	1 month	77,589	96,980	2.50	5665	77.1%	7.6

Table 2: Graphs used in our experiments. LCC stands for the largest connected component. Average Distance is the average number of hops between two randomly selected nodes.

algorithms. They are as follows: **Oregon Autonomous System (AS)**:[4] There are four graphs in this dataset, each collected in a week. The data was collected in 2001. A node represents an autonomous system. An edge is a connection inferred from the Oregon route-views. **Yahoo! Instant Messenger (YIM)**:[5] YIM is a online chatting program. A node is a Yahoo! IM user. An edge indicates a communication between two users. We have three social graphs here, each corresponding to a day's worth of data. The data was collected in April 2008. We use the data in the 1^{st}, 11^{th}, and 21^{st} day of the month. **Facebook user-postings (FB)**:[6] We have five social graphs here, each corresponding to a day's worth of data. The data was collected in March 2013. We use the data from the 10^{th} to 14^{th} day of the month. A node is a Facebook user. There is an edge between two users if there is a "posting" event between them in the selected day. **Twitter re-tweet (TT)**:[7] We have four social graphs, each corresponding to a month's worth of data. The data was collected from May to August 2009. A node represents a Twitter user. There is an edge between two users if there is a re-tweet event between them.

4.2 Experimental Setup We compare *MET* with the following nine methods. **(1)** *MET-Naive*, see Section 3 for details. **(2)** *NetMelt*, where edges are selected based on eigenscores of the leading eigenvalue [12]. **(3)** *NetMelt*$^+$, where eigenscores are recomputed after edge edge deletion; it is an improved but much slower version of *NetMelt*. **(4)** *Rand*, where edges are selected randomly for deletion. **(5)** *Rich-Rich*, where edge are selected based on the highest $d_{src} \times d_{dst}$; d_{src} and d_{dst} are the degrees of the source and destination nodes [4]. **(6)** *Rich-Poor*, where edges are selected based on the highest $|d_{src} - d_{dst}|$. **(7)** *Poor-Poor*, where edges are selected base on the lowest $d_{src} \times d_{dst}$. **(8)** *EBC*, where edges are selected based on the highest edge betweenness centrality, which is defined as the number of shortest paths from all pairs of vertices that pass through an edge. **(9)** *Miobi*, where edges are selected to minimize the robustness of the graph, which is defined as the average of the top K eigenvalues of the adjacency matrix [4].

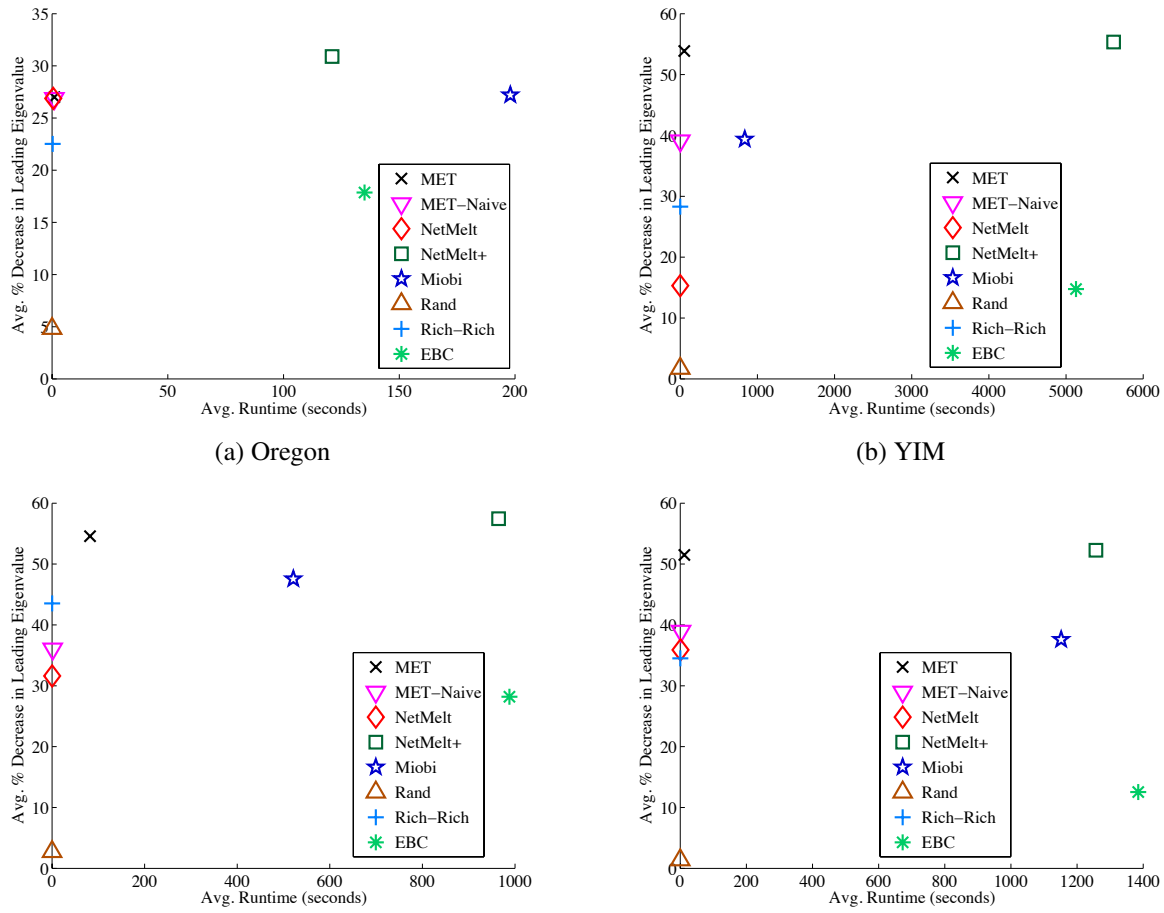
In *MET-Naive*, the default value for T is 10 (unless otherwise noted). *MET* does not need the parameter T to be specified, because it adaptively selects T. The default value for the slack parameter $epsilon$ is 0.5. We evaluate the effect of selecting different values of $epsilon$ in Figure 3.

One of our evaluation criteria is *efficacy*, which is the relative drop in the leading eigenvalue: $\%drop = 100\frac{\lambda_1 - \bar{\lambda}_1}{\lambda_1}$, where λ_1 is the leading eigenvalue of the original graph and $\bar{\lambda}_1$ is the leading eigenvalue of the graph after removing k edges. The higher the percentage drop in the leading eigenvalue, the better the efficacy.

We also evaluate the *efficiency* of the aforementioned algorithms by measuring the wall clock time (in seconds). Obviously, lower runtime is better. All experiments were conducted on a Macbook Pro with CPU 2.66 GHz, Intel Core i7, RAM 8 GB DDR3, hard drive 500 GB SSD, and OS X 10.8. The code is written in Matlab.

For brevity, we omit the results on *Rich-Poor* and *Poor-*

[4] http://topology.eecs.umich.edu/data.html
[5] http://webscope.sandbox.yahoo.com
[6] Proprietary data given to us by a collaborator.
[7] http://socialcomputing.asu.edu



(a) Oregon (b) YIM

(c) FB (d) TT

Figure 1: (Best viewed in color.) Trade-off between efficacy and efficiency when removing 1000 edges from our graphs. Higher percentage drop in the leading eigenvalue and lower runtime is better (i.e., the upper left corner of the plots). *MET* and *NetMelt*$^+$ achieve similar percentages of drop in the leading eigenvalue, but *NetMelt*$^+$ is much slower than *MET*. For example, *NetMelt*$^+$ is 100 times slower than *MET* on YIM graphs.

Poor. In all of our graphs, they performed poorly when compared to *Rich-Rich*. In addition, we omit the results on *Miobi* (with re-computation after 50 edge-deletions as recommended in [4]). *Miobi* (with re-computation) achieved lower efficacy than *MET* and *NetMelt*$^+$, while its runtime was higher than *NetMelt*$^+$.

4.3 Results

4.3.1 Trade-off between Efficacy and Efficiency Figure 1 shows the trade-off between the percentage drop in λ_1 (i.e., efficacy) and wall-clock runtime (i.e., efficiency).[8] *NetMelt* is a fast algorithm since it computes the eigenscores only once, but it performs poorly in social graphs where the eigen-gap is small. *NetMelt*$^+$ is very slow because it computes the eigenscores after every edge deletion. *MET* achieves similar efficacy to *NetMelt*$^+$, while having a low runtime. For example, *MET* is 108 times faster than *NetMelt*$^+$ on the Yahoo! IM graphs, while preserving 99% of *NetMelt*$^+$'s efficacy. On the FB graphs, *MET* is 12 times faster than *NetMelt*$^+$, while preserving 96% of *NetMelt*$^+$'s efficacy. On the TT graphs, *MET* is 30 times faster than *NetMelt*$^+$, while preserving 99.3% of *NetMelt*$^+$'s efficacy.

Table 3 lists the average values for the number of re-computations of eigenscores and the number of eigenvalues tracked by *MET*. We observe that *MET* needs to track a few eigenvalues with a small number of re-computations, which makes *MET* much faster than *NetMelt*$^+$. On the FB

[8] For brevity, we have omitted the efficacy results as the budget k varies.

Dataset	Avg. # of eigenvalues being tracked	Avg. # of re-computations
Oregon	2.08	1.25
YIM	5.17	10.3
FB	6.83	24.6
TT	3.03	9.4

Table 3: Average number of eigenvalues being tracked in *MET* and the average number of times *MET* re-computes eigenscores, when removing $k = 1000$ edges from each graph. *MET* tracks small numbers of eigenvalues and seldomly does re-computation, which makes *MET* a fast and effective algorithm.

graphs, *MET* needs to track more eigenvalues and do more re-computations than on YIM and TT graphs. The reason for this is because the eigen-gaps on the FB graphs are very small compared to those on the YIM and TT graphs (see Figure 2).

4.3.2 Effectiveness of the Slack Variable ϵ in *MET* Figure 2 plots the top eigenvalues of (*i*) the original graph and (*ii*) after *MET* removes 1000 edges. The gap between λ_1 and λ_{21} is as small as 0.37 on Yahoo! IM, 0.36 on TT, and 0.05 on FB. When this situation happens, the (eigenscore) re-computation condition is triggered more often (i.e., $max(estimated\ \lambda_i) < \lambda_T$ after few edge deletions). One possible solution is to track even more eigenvalues, which also increases runtime. *MET* enables trade-off between efficacy and runtime by introducing a slack variable ϵ. Figure 3 shows the effect of increasing ϵ. In real-world applications, one may allow a small drop in the efficacy to improve the runtime. To achieve the highest efficacy, we simply set ϵ to 0. In this case, *MET* can preserve more than 99% of efficacy compared with *NetMelt*$^+$ in all social graphs.

4.3.3 Simulating Virus Propagation As mentioned previously, the leading eigenvalue λ_1 of the adjacency matrix affects the dissemination of a virus. We simulate the popular Susceptible-Infected-Susceptible (SIS) model in order to evaluate the efficacy of minimizing the virus by different methods. In the SIS model, a node can be in one of two states: susceptible or infected. A susceptible node will be infected at some infection rate β if it connects with infected neighbors. In the mean time, an infected node also tries to recover by itself at a rate δ. In our experiment, we removed $k = 1000$ edges from the original graph and evaluated how many nodes were infected. We ran 100 experiments for 100 time steps and report the average results. In Figure 4, the x-axis is the time step and the y-axis is the fraction of nodes infected at a particular time step. The lower the fraction of infected nodes, the better. We observe that *MET* always achieves similar performance to *NetMelt*$^+$; but recall that *MET* is substantially faster than *NetMelt*$^+$.

4.4 Discussion The runtime of *MET* is $100+$ times faster than *NetMelt+* on Yahoo IM!, 30 times faster on TT, but only $10+$ times faster on FB. The top eigenvalues of FB become very close to each other after removing k edges. For example, Figure 2 showed that the difference between λ_1 and λ_{21} in FB can be as small as 0.05. When the top-T eigenvalues are so close (as in FB), *MET* has to track more eigenvalues or re-compute the eigenscores more often, both of which lead to higher runtimes. In real-world applications, one may allow a small loss in efficacy by using *MET*'s slack variable ϵ, which makes *MET* faster. For example, on FB, *MET* is 12 times faster than *NetMelt*$^+$ while preserving 96% of *NetMelt*$^+$'s efficacy.

A key question is: *How far are the MET results from the upper-bound on the optimal solution that minimizes the largest eigenvalue of a graph?* As mentioned previously, finding the optimal solution is NP hard. To answer this question, we show the upper-bound on the largest degree d_{max} of a graph after removing k edges.This result will help us provide an upper-bound on the drop in the leading eigenvalue in the optimal solution. To the best of our knowledge, ours is a tighter upper-bound on the optimal solution than previously published. Before moving forward, here are a couple of preliminaries: (1) As shown in [11], $\Delta\lambda_i = \lambda_i - \bar{\lambda}_i \approx getEigenscore(e, \lambda_i, \mathbf{u}_i, \mathbf{v}_i)$. Section 2 defined $getEigenScore(e, \lambda_i, \mathbf{u}_i, \mathbf{v}_i)$. As the name suggests, it returns edge e's eigenscore associated with λ_i. (2) $\sqrt{d_{max}} \leq \lambda_1$, where d_{max} is the maximum degree of G. If $\mathbf{A}'$ is a sub-matrix of $\mathbf{A}$, according to Proposition 3.1.1 in [2]), we have:

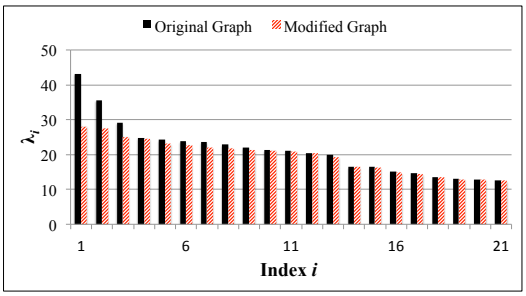
$$\lambda_1(\mathbf{A}') \leq \lambda_1(\mathbf{A}) \tag{4.1}$$

Then, from the definition of λ, we can write:

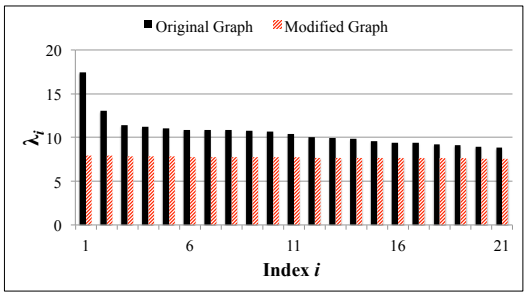
$$\lambda_1(\mathbf{A}) = \max_{\mathbf{x}} \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} \tag{4.2}$$

If we set $\mathbf{x}$ in Eq. 4.2 to be a vector of all ones except for the highest degree node, where it will have $\sqrt{d_{max}}$, then we observe that $\sqrt{d_{max}}$ is an eigenvalue of a star graph with the highest degree of d_{max}. Putting Eq. 4.1 and Eq. 4.2 together, we get $\sqrt{d_{max}} \leq \lambda_1$.

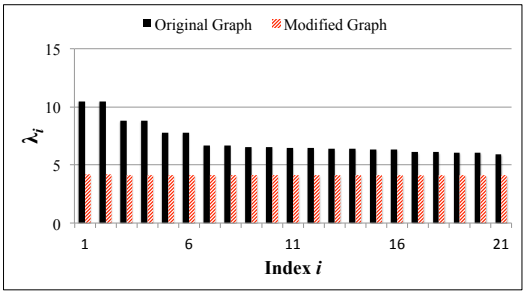
Now suppose we have a sequence of numbers: $d_1 \geq d_2 \geq ... \geq d_n$. This is not a degree sequence, just a sequence of numbers. Consider the trivial problem of minimizing the maximum element in this sequence by reducing one element at a time. Obviously, the best way of accomplishing this is to reduce d_1 until it is equal to d_2, then reduce d_1 and d_2 alternately until they are equal to d_3, then reduce d_1, d_2, and d_3 alternatively until they are equal to d_4, and so on. Suppose



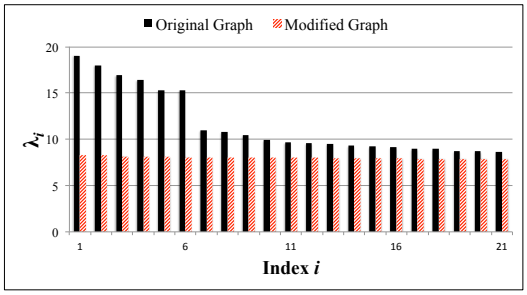
(a) λ's of Oregon-1



(b) λ's of YIM-1



(c) λ's of FB-1



(c) λ's of TT-1

Figure 2: Top eigenvalues of the original graph and the modified graph (i.e., the graph after removing 1000 edges selected by *MET*). The top eigenvalues in social networks are very close to each other after removing 1000 edges with *MET*. After the edge removal, the gaps between λ_1 and λ_{21} in *YIM-1*, *FB-1*, and *TT-1* are 0.37, 0.05, and 0.36, respectively.

$f(x)$ is defined to be the maximum value of this sequence after performing x such reductions in this way.

In the case of a graph, if we are allowed to remove k edges in a given graph, the maximum degree of the modified graph can not be smaller than $f(2k)$. We can find the upper-bound on the percentage drop in the leading eigenvalue in Eq. 4.3. Notice that $0 < \sqrt{f(2k)} \leq \bar{\lambda}_1 \leq \lambda_1$, where $\bar{\lambda}_1$ is the leading eigenvalue after removing edges.

$$\%drop = \frac{100(\lambda_1 - \bar{\lambda}_1)}{\lambda_1} \leq \frac{100(\lambda_1 - \sqrt{f(2k)})}{\lambda_1} \tag{4.3}$$

Figure 5 plots the percentage drop of the leading eigenvalue when using *MET*, using $\sqrt{f(2k)}$, and using $d_{max} - k$. Our $\sqrt{f(2k)}$ provides a tighter upper-bound on the optimal solution than $d_{max} - k$, which was introduced in [13, 12].

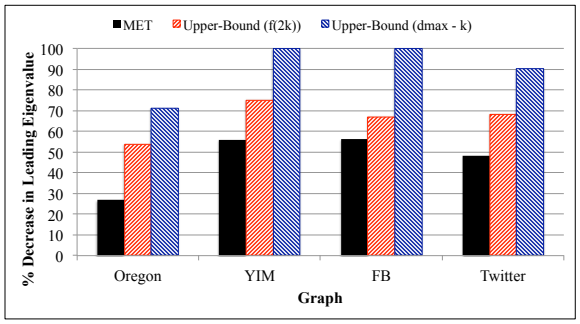
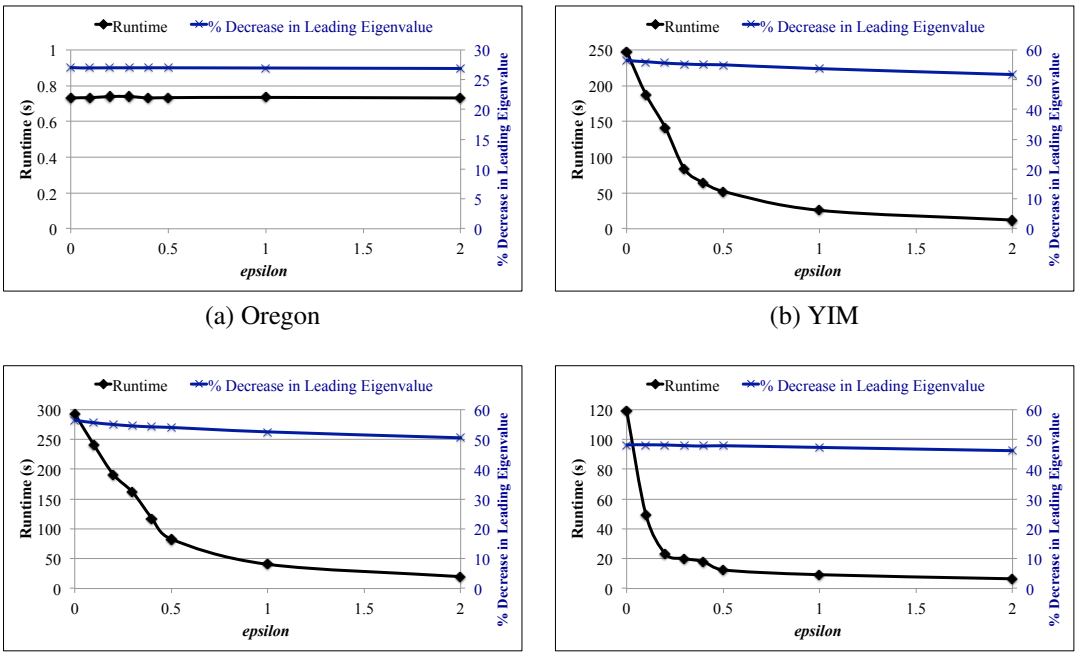


Figure 5: Upper-bound on the percentage decrease in λ_1 with the optimal solution using the highest node-degree of the graph. Budget $k = 1000$. The optimal solution must be in between the black bars (*MET*) and red bars (our upper-bound based on $\sqrt{f(2k)}$). The red bars provide a tighter bound than the blue bars (which define the upper-bound based on $d_{max} - k$).

5 Related Work

The closet research to *MET* fall into two categories: (1) controlling dissemination by deleting edges and (2) spectra of graphs.

Controlling dissemination by deleting edges. Tong *et al*. [12] use the eigenscores associated with the leading eigenvalue λ_1 to select the "best" edges (i.e., ones that yield the highest drop in λ_1) for deletion. Their method, called



(a) Oregon (b) YIM

(c) FB (c) TT

Figure 3: The effects of selecting ϵ values on percentage drop in λ_1 (i.e., efficacy) and runtime in *MET*. The values on the y-axes are the averages over graphs of the same type. Smaller ϵ values create higher eigen-drops but also higher runtimes. Note that when ϵ is 0, the runtimes are still lower than *NetMelt+*. That is, *MET* is 23 times faster in Yahoo! IM, 4 times faster in FB, and 11 times faster in TT; while preserving 99.9%, 99.2%, and 99.9% efficacy, respectively. The performance of *MET* on Oregon is not affected by the value of ϵ due to the large eigen-gaps in Oregon (i.e., ϵ does not trigger re-computation).

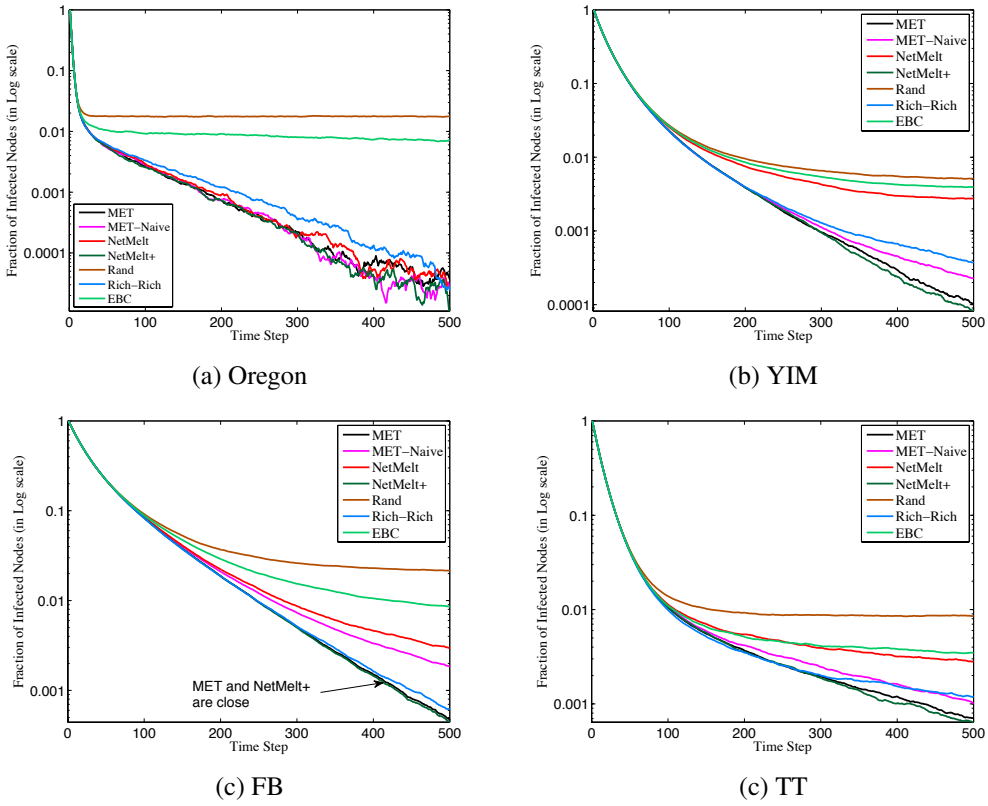
NetMelt tracks only λ_1; thus its performance is hindered on a graph with small eigen-gaps (see Section 4). Bonchi [1] and Goyal *et al*. [6] identify "important" links (i.e., ones that most likely explain propagation in a social network) by studying past propagation logs. They examine the influence probabilities from available propagation events since the social-network connections are not associated with probabilities. These influence probabilities are dynamic and can change over time. Our work uses the topology of the graph, rather than propagation logs. Kuhlman *et al*. [8] study blocking simple and complex contagions via edge removal in ratcheted dynamical systems, where infected nodes cannot recover. Nodes are infected by contacting one infected node in the simple-contagion scenario and at least two infected nodes in the complex-contagion scenario. They investigate various heuristics for this NP-hard problem on weighted and unweighted networks. *MET* is agnostic to the recovery of a node after infection. Chan *et al*. [4] track multiple eigenvalues for the task of manipulating network robustness. We compare *MET* with their approach in Section 4. *MET* has better efficacy and efficiency than their approach.

Spectra of a graph is about the relationship between the topology and the eigen decomposition of the graph. In particular, spectra of a graph provides information about the graph's connectivity, robustness, and randomness [5, 2]. Stewart and Sun [11] present a very nice study of graph spectra under perturbation.

6 Conclusions

We observed that small eigen-gaps are prevalent in social graphs, which make the problem of minimizing dissemination by deleting edges a challenge for approaches that only track the leading eigenvalue or perform eigen decomposition only once. We introduced *MET*, an efficient and effective approach that minimizes dissemination on a graph, regardless of its eigen-gaps. *MET* deletes k edges by tracking multiple eigenvalues adaptively and by re-computing eigenscores when necessary. We evaluated *MET* against several competing methods on various large graphs and showed the efficacy and efficiency of *MET*. We found that *MET* yields the best combinations of efficacy and efficiency. Moreover, we showed how far the *MET* results are from the upper-bound on the optimal solution.

Future work. We will investigate algorithms that can minimize dissemination on a graph while keeping its community structure intact.



(a) Oregon (b) YIM

(c) FB (c) TT

Figure 4: (Best viewed in color.) Comparing the capability of different methods in minimizing the number of infected nodes in the SIS model. Lower on the y-axis is better. We removed $k = 1000$ edges from each graph using different algorithms, ran 100 simulations and took the average number of infected nodes. The normalized virus strengths of Oregon, YIM, FB, TT graphs are 1.43, 2.49, 2.34, 2.07, respectively. The virus strength is $\frac{\beta}{\delta}$, where β, δ are the infection and death rates of the virus. Our method *MET* achieves similar performance to *NetMelt*$^+$ in minimizing the number of infected node, but is much faster than *NetMelt*$^+$. *MET* always beats the other methods in social graphs, which often have small eigen-gaps.

References

[1] F. Bonchi. Influence propagation in social networks: A data mining perspective. In *IEEE Intelligent Informatics Bulletin*, page 2, 2011.

[2] A. Brouwer and W. H. Haemers. *Spectral of Graph*. Springer, 2009.

[3] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur.*, 10(4):1:1–1:26, 2008.

[4] H. Chan, L. Akoglu, and H. Tong. Make it or break it: Manipulating robustness in large networks. In *SDM*, pages 325–333, 2014.

[5] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[6] A. Goyal, F. Bonchi, L. V. S. Lakshmanan, and S. Venkatasubramanian. On minimizing budget and time in influence propagation over social networks. *Social Netw. Analys. Mining*, 3:179–192, 2013.

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

[8] C. J. Kuhlman, G. Tuli, S. Swarup, M. V. Marathe, and S. S. Ravi. Blocking simple and complex contagion by edge removal. In *ICDM*, pages 399–408, 2013.

[9] C. Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral. *J. Res. Nat. Bur. Stand.*, pages 255–282, 1950.

[10] B. A. Prakash, D. Chakrabarti, M. Faloutsos, N. Valler, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. In *ICDM*, pages 537–546, 2011.

[11] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[12] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM*, pages 245–254, 2012.

[13] H. Tong, B. A. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. H. Chau. On the vulnerability of large graphs. In *ICDM*, pages 1091–1096, 2010.