Noname manuscript No.
(will be inserted by the editor)

Assessing and Altering Robustness of Large Graphs

Hau Chan · Leman Akoglu · Hanghang Tong

Received: date / Accepted: date

Abstract The function and performance of many networked systems, such as communication and transportation networks, rely on their resilience, defined as their ability to continue functioning in the face of damage to parts of the network. The damage can be in the form of intentional targeted attacks or failures of random or cascading nature. Prior research has proposed various measures to assess graph robustness as well as various manipulation strategies to alter it. In this work, our contributions are two-fold. First, we critically analyze a diverse list of proposed robustness measures and identify their strengths and weaknesses in quantifying graph robustness. Our analysis suggests natural connectivity, based on the weighted count of closed walks in a graph, to be a reliable measure. Second, we formulate three graph manipulation problems involving node and edge deletions to degrade, and edge additions to improve the robustness of a given graph as defined by its natural connectivity. We study the hardness of these problems and propose the *first* principled alteration algorithms that directly optimize the corresponding robustness measure. Our direct optimization leads to significant improvement over many existing ad-hoc heuristic strategies. Extensive experiments on real-world datasets demonstrate the effectiveness and scalability of our methods against a long list of competitor heuristics.

Keywords graph mining · robustness · vulnerability assessment · connectivity · graph spectrum · edge manipulation · node manipulation · network design

Hau Chan
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794.
Tel.: +1-631-632-9801, Fax: +1-631-632-2303.
E-mail: hauchan@cs.stonybrook.edu

Leman Akoglu
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794.
E-mail: leman@cs.stonybrook.edu

Hanghang Tong
Department of Computer Science, City College, City University of New York, New York, NY 10031 USA.
E-mail: tong@cs.ccny.cuny.edu

1 Introduction

Robustness, generally speaking, measures the resilience of a network in response to the external perturbations (e.g., intentional attacks or random failures), is a fundamental property for a variety of networks, such as social, information, communication, biological networks and so on. Networks that sustain their functionality and responsiveness under such changes (targeted or random) are considered to be more robust than others that fail to do so.

In the past few decades, research on robustness has been concerned with *measuring* the robustness of a given network [1,16,15], *tracking* its dynamics when the network evolves over time [35], *manipulating* its structure (e.g., by removing nodes) to alter its robustness [1,3,19], and *comparing* the robustness of different networks under a certain type of perturbation [1,7,4].

In particular, a vast majority of prior works have focused on quantifying robustness and thanks to those efforts we now have a variety of different robustness measures, e.g., size of largest connected component, inverse shortest distances, algebraic connectivity, etc. In principle, the common ingredient/component of these measures is the level of network *connectivity*. While each of these robustness measures has its own emphasis and rationality, as we discuss in the next section, many measures have several shortcomings in capturing the desired connectivity and resilience properties of networks. For example measures based on shortest distances, such as diameter, efficiency, etc., are quite sensitive to small alterations. Moreover, they do not account for redundancy, i.e., alternative paths between the nodes. Several other measures, such as algebraic connectivity, clustering coefficient, etc., do not change strictly monotonically when e.g., new edges are added to the network. In addition, some measures including algebraic connectivity and diameter, are meaningless for disconnected graphs; they either take the same value for all such graphs or are not well-defined (see §2.1 for more details).

Ideally, a fully connected network is the most robust; however it is not feasible to design fully connected real-world networks due to several constraints, such as physical space, budget, etc. Alternatively, building alternative (i.e., redundant) paths among the nodes helps improve the resilience against damage in the network. The more and shorter these alternative paths are, the better the resilience would be. Thus, several more recent robustness measures [26,45] are built on the so-called *subgraph centrality*, which measures the total (weighted) count of loops in the network.

While most prior research focused on robustness measures, little has been done on *how to manipulate the robustness of a given network by modifying its underlying link structure*. For instance, how can we *enhance* the robustness of a power grid network by carefully introducing a few new power lines? How can we *break down* a disease network by removing (say by an operation) some of its cells? How can we maximally break down an adversary network (e.g., a terrorist network) by cutting out some of its communication channels? To date, little, except a few ad-hoc solutions has been proposed to answer these kinds of questions (see §2.2 for more details).

In this work, we focus on two critical problems related to network robustness; (1) quantifying/measuring, and (2) manipulating/altering the robustness in large networks. In particular, we address the following questions: (Q1) *Robustness measure*:

While there exist many different robustness measures, it is unclear for the practitioner which measure should s/he choose. What is a good robustness measure that captures the desired resilience properties of a graph? (Q2) *Manipulation algorithms*: Given such a desired measure, how can we design effective and scalable algorithms that directly optimize it for manipulating robustness?

We start by carefully choosing *natural connectivity* [45] as a reliable robustness measure. Next we propose a novel framework called MIOBI (for Make It or Break It) for controlling the robustness of a given graph by modifying its topology. In particular, we address two new problems. First, we focus on the problem of maximally decreasing the robustness of a given network by deleting nodes or links. Second, we study the problem of maximally increasing the robustness by carefully introducing a set of new links. A unique feature of our methods is that they aim to *directly* optimize the corresponding robustness measure, which leads to significant performance improvement over the existing, ad-hoc solutions. The proposed methods scale to large graphs, with near-linear complexity in time and space. We summarize our main contributions as follows:

– *Analysis of robustness measures*: We analyze several measures in the literature for their capabilities of capturing desired resilience properties of graphs, such as accounting for alternative paths (redundancy), strictly increasing with addition of new edges (strict monotonicity), etc. We conclude that natural connectivity, that satisfies all of our criteria, proves to be a reliable measure.
– *Robustness manipulation problems*: We formulate two manipulation problems to degrade graph robustness via (1) edge and (2) node removal (respectively called MIOBI-BREAKEDGEand MIOBI-BREAKNODE), as well as one problem to improve robustness via edge addition (called MIOBI-MAKEEDGE). We further analyze the computational hardness of the posed problems theoretically.
– *Principled robustness manipulation algorithms*: We propose effective and scalable algorithms to identify the best operations for a given budget for each problem. Our algorithms are based on theoretical bases and provide the *first* principled, rather than ad-hoc, solutions. Prior research has considerable discrepancy between robustness measures used and manipulation algorithms employed (e.g., largest connected component size as measure vs. degree-based removal for manipulation). We bridge this gap by *directly* optimizing our chosen robustness measure under manipulation.
– *Extensive experiments*: We evaluate our methods on a long list of real-world datasets across various domains (e.g., email, P2P, Internet AS topology) for effectiveness and scalability. We show that i) our proposed methods outperform a long list of ad-hoc strategies, and ii) successfully scale to large-scale graphs with the empirical running time growing near-linearly in graph size (number of edges).

2 Related Work

Due to its wide range of applications, robustness has been studied extensively in various communities, including physics, biology, networking, mathematics, and computer science. We organize related work on robustness into two sections: (1) work on proposing measures to quantify robustness, and (2) work on studying the effects of network manipulation on robustness.

2.1 Measuring Robustness

Simple and effective measures of robustness are essential in the areas of network design and monitoring. In graph theory, robustness can comprise of properties ranging from redundancy and diversity, to concepts such as the ability to operate under perturbation or the efficiency of feedback mechanisms. In principle, the graph *connectivity* is a fundamental measure of the robustness of a network.

In [1] network robustness is defined as the critical removal fraction of nodes (or edges) from the network that causes its sudden disintegration. To monitor disintegration, they propose to track the diameter, relative size of the largest connected component (LCC), and average size of isolated clusters. Intuitively, as the fraction of removed nodes/edges increases, the performance of the network eventually collapses at a critical fraction f that corresponds to the network robustness; the larger f is, the more robust the network is. The critical removal fraction f, however, while can be computed analytically for special network structures [7,4], in general it needs to be computed via simulations. Moreover, component sizes do not fully reflect the *level* of connectedness of a given network. Pairwise connectivity [39], based on the fraction of pairs that have at least one path between them, has also been used to quantify the connectedness of a graph. Like LCC size, this measure also assesses connectivity at a coarse level, ignoring the count and length of (back-up) paths between the node pairs.

Other prior works have proposed mathematically compact representations to quantify robustness. These measures include connectivity based on minimum node/edge cut [16], average inverse shortest path (geodesic) distances of connected components [1,3,19], and algebraic connectivity based on the second smallest (or first non-zero) eigenvalue of the Laplacian [15]. However, these measures only partly reflect the ability of graphs to retain connectedness after manipulation, and fail to exhibit the variation of robustness sensitively [45]. In particular, shortest paths are prone to change drastically with simple alterations and do not capture redundancies (i.e., alternative paths). Moreover, algebraic connectivity takes the value of zero for all disconnected graphs which makes it a measure that is too coarse for complex networks, and it does not change monotonically with the addition/deletion of more edges [45].

Related to geodesic distance, [35] propose a modified measure for time-varying graphs called shortest temporal distance. [11,26] incorporate the spectral gap related to spectral expansion properties [12] as well as subgraph centralities of the nodes in the network, to quantify network robustness. More recently, [31] propose a community-centric measure of robustness where they quantify the the change in the clustering structure of the topology as measured by modularity [30].

Other measures include toughness [6], scattering number [20], tenacity [25], integrity [2], fault diameter [22], restricted connectivity [5], and isoperimetric number (related to node/edge expansion) [29]. These take into account the cost as well as the magnitude of damage to a network. However, these are combinatorial measures which are often hard to compute efficiently for general graphs.

In summary, while all these and several other measures capture graph connectivity one way or another, they have one or more of the following shortcomings: i) prone to drastic changes by small graph alterations, ii) partially capturing connectivity or alternative paths, iii) combinatorial to compute efficiently, iv) meaningful only for connected graphs, and v) non-monotonically changing by more modifications. Thus, we adopt *natural connectivity* [45] as our measure, which successfully avoids these pitfalls (details in §3).

2.2 Altering Robustness

One can study the change in robustness under i) few and random node/edge failures, or ii) many or targeted failures (or attacks). In their study, [1] showed that scale-free graphs are resilient to random failures but sensitive to targeted attacks, while for random networks there is smaller difference between the two. As such, researchers proposed and studied different manipulation strategies for targeted attack scenarios for real-world networks.

The most frequently studied strategy to *degrade* robustness has been the removal of most connected (i.e., highest degree) nodes [1,3,19]. Further, [19] compared this strategy to node/edge removals based on betweenness centrality and showed that betweenness yields better results, especially for removal of edges. Different from most, and similar to our MIOBI-MAKEEDGE, [3] investigated modification schemes to *improve* network robustness. In particular, they studied (1) edge rewiring, and (2) edge addition strategies based on i) random, or ii) preferential schemes. They concluded that in general preferential edge additions, i.e. connecting lowest degree nodes, yield the best result. Other works also explore edge rewiring strategies to improve graph robustness [24,28,36].

More theoretical work in this area includes [21] that define what is called an (ϵ, k)-failure where deletion of k nodes/edges disconnects a graph of n nodes into two components, each of at least size ϵn. They give theoretical bounds on the existence and selection of a set of "witness" nodes D called detection nodes, such that some two nodes in D gets disconnected when/if an ϵ, k-failure occurs. As such, this work focuses on the detectability of failures/attacks.

Finally, designing networks that are optimal with respect to some survivability criteria [16,38,34] is a related but different research topic. These consider building a network from scratch, whereas we aim at modifying an existing network as effectively as possible under a given budget, without causing substantial changes to its existing structure.

We remark that a vast majority of prior research revolves around ad-hoc manipulation techniques. They either use simulations, assume special network models/structures (e.g., random graphs), or develop heuristic edge/node elimination strategies, and compare them across each other (e.g., betweenness versus degree based removals). There exists a small group of works which propose systematic approximation algorithms to manipulate a graph robustness measure of their interest. These include the optimization for pairwise connectivity [9] and community-structure [31]. Our work is in the same lines of this latter group, where we propose a set of *principled*, rather than ad-hoc, algorithms for graph manipulation.

3 Assessing Graph Robustness

In this section we first introduce the notation and then describe natural connectivity, the robustness measure we adopt in our work.

3.1 Notation

Table 1 lists the notation used in text. We consider undirected unipartite irreducible graphs $G(V,E)$ with a vertex/node set V of size n and an edge set E of size m. An (undirected) edge of G between nodes p and r in V is written as $(p,r) \in E$. The set of neighbors and degree of a node $i \in V$ are denoted by $\mathcal{N}(i)$ and d_i, respectively. We use uppercase bold letters for matrices (e.g., $\mathbf{A}$) and lowercase bold for vectors (e.g., $\mathbf{a}$). Given a matrix $\mathbf{A}$, $\mathbf{A}(i,j)$ corresponds to the element at i^{th} row and j^{th} column of $\mathbf{A}$. Moreover, $\mathbf{A}(i,:)$ and $\mathbf{A}(:,j)$ represent the i^{th} row and j^{th} column of matrix $\mathbf{A}$, respectively. We denote the transpose with a prime (e.g., $\mathbf{A}'$, $\mathbf{a}'$). The eigenvalue and associated eigenvector pairs of the adjacency matrix $\mathbf{A}$ are denoted by $(\lambda_j, \mathbf{u_j})$. The i^{th} element of an eigenvector $\mathbf{u_j}$ is represented by $\mathbf{u_{ij}}$.

Table 1: Notation used throughout text.

Symbol	Definition and Description
$G(V,E)$	undirected, unipartite network
n	number of nodes $\lvert V\rvert$ in the network
m	number of edges $\lvert E\rvert$ in the network
k	budget (number of edges/nodes to add/delete)
(p,r)	undirected edge in E between nodes p and r
$\mathbf{A},\mathbf{B},\ldots$	matrices (bold upper case)
$\mathbf{A}(i,j)$	element at i^{th} row and j^{th} column of $\mathbf{A}$
$\mathbf{A}(i,:)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:,j)$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{a},\mathbf{b},\ldots$	vectors (bold lower case)
$\mathcal{A},\mathcal{B},\mathcal{C},\ldots$	sets (calligraphic)
λ_j	j^{th} largest (in module) eigenvalue of $\mathbf{A}$
$\mathbf{u_j}$	$n \times 1$ eigenvector of $\mathbf{A}$ associated with λ_j.
$\mathbf{u_{ij}}$	i^{th} entry in $\mathbf{u_j}$
$\mathcal{N}(i)$	set of neighbors of node i
d_i	degree of node i

3.2 Our Robustness Measure

In this paper, we adopt a spectral measure of robustness in complex networks, called *natural connectivity* [45], which can be written as follows.

$$\bar{\lambda}^{\mathbf{A}} = \ln(\frac{1}{n}\sum_{j=1}^{n} e^{\lambda_j}) \tag{1}$$

which corresponds to an "average" eigenvalue of $G(V, E)$, where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ denote a non-increasing ordering of the eigenvalues of its adjacency matrix $\mathbf{A}$. When the context is clear, the superscript $\mathbf{A}$ will be omitted (i.e. $\bar{\lambda}^{\mathbf{A}} \equiv \bar{\lambda}$).

Natural connectivity not only has a simple mathematical formulation that can be interpreted as the average eigenvalue of the graph, but it also has clear physical and structural meaning that can be tied to several connectivity properties of networks. In particular, it explicitly characterizes the redundancy of alternative paths in the network by quantifying the weighted number of closed walks of all lengths.

A walk in G is an alternating sequence of nodes and edges $v_0e_1v_1e_2v_2\ldots e_kv_k$ where $v_i \in V$ and $e_i(v_{i-1}, v_i) \in E$. The walk is closed if $v_0 = v_k$. The number of walks is an important measure for network robustness. Intuitively, it captures the redundancy of routes between the nodes and redundant routes ensure that connections between nodes remain possible in face of damage to the network. Ideally, robustness could consider the number of alternative routes of different lengths for all pairs of nodes, however this measure becomes intractable for very large graphs. Therefore, natural connectivity focuses on the closed walks of the graph.

Closed walks can be directly related to the subgraphs of a graph and derived from the *sum* of the subgraph centralities of all the nodes in the graph. The subgraph centrality $SC(i)$ of a node i is determined based on the "weighted" sum of the number of closed walks that it participates in. Therefore,

$$\begin{aligned} S(G) &= \sum_{i=1}^{n} SC(i) = \sum_{i=1}^{n}\sum_{k=0}^{\infty}\frac{(\mathbf{A}^k)_{ii}}{k!} = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{u_{ij}}^2 e^{\lambda_j} \\ &= \sum_{j=1}^{n} e^{\lambda_j}\sum_{i=1}^{n}\mathbf{u_{ij}}^2 = \sum_{j=1}^{n} e^{\lambda_j} \end{aligned}$$

where $(A^k)_{ii}$ is the number of closed walks of length k of node i. The $k!$ scaling ensures that (i) the weighted sum does not diverge, and (ii) longer walks count less. We note that $S(G)$ is also known as the Estrada index of the graph [10]. As such, we can write

$$\bar{\lambda} = \ln(\frac{1}{n}\sum_{j=1}^{n} e^{\lambda_j}) = \ln(\frac{1}{n}S(G))$$

Moreover, natural connectivity is closely related to *self-communicability* [13] of nodes in the network. The general communicability function between nodes p, q is written as

$$C_{pq} = \sum_{k=0}^{\infty} c_k(\mathbf{A}^k)_{pq}$$

and thus, subgraph centrality can be thought of as self-communicability with *factorial penalization* of walk lengths. The general communicability between any pair of nodes p, q (again with factorial penalty) can be written as (using Taylor series and the spectral decomposition of $\mathbf{A}$)

$$C_{pq} = \sum_{j=1}^{n} \mathbf{u_{pj}} \mathbf{u_{qj}} e^{\lambda_j}.$$

The above arguments show that (1) natural connectivity exhibits characteristics about the communicability in the network through alternative paths, which closely relate to robustness. It associates the robustness to network topology, graph spectra, and dynamical properties. Moreover, it was shown [37] that (2) natural connectivity has strong discrimination in quantifying the robustness of complex networks and can exhibit the variation of robustness sensitively even for disconnected networks (unlike e.g., algebraic connectivity). Finally, (3) natural connectivity changes strictly monotonically with the addition/deletion of more and more nodes/edges [45], which agrees with intuition (unlike e.g., node/edge connectivity, algebraic connectivity). These indicate that the natural connectivity can measure the robustness of complex networks stably even for very small sized and disconnected networks. For these reasons, we adopt natural connectivity as our network robustness measure in our study.

4 Altering Graph Robustness

Given a network and its robustness, a simple manipulation technique to increase or decrease its robustness is to add (new) edges or remove (existing) edges/nodes, respectively. In this paper, we will focus on node/edge deletion and edge addition operations to manipulate graph robustness.

4.1 Problem Definitions

We start by introducing the problems we address to manipulate network robustness.

4.1.1 Decrease the Robustness of the Network

There are many reasons why one would want to decrease the robustness of the given network. This depends on the motivations and objectives of the manipulator(s).

For example, if an adversary wants to attack the (cyber or physical) network, of course, he/she would want to target/destroy some of the edges and the nodes of the network such that they will maximize some utility function. Understanding the strategies of the attacker would help the law-enforcers or network designers to better protect the network.

On the other end of the spectrum, the goal of the network designers or the law-enforcers is completely opposite of the attacker yet there are times in which the designers want to break the robustness of the network. Consider a scenario in which there is an epidemic (a virus in a computer network or a disease in a human network). To prevent the spreading the epidemic, the designers may shut down some links or even some nodes in the network temporary. The law-enforcers might want to break the terrorists communicate network and target the "important" communication channels (edges) and terrorists (nodes) such that they will break the network the most.

To make the network less robust (or decrease the robustness of the network), we consider the network manipulations in which one can (a) remove some number of (existing) edges and (b) remove some number of (existing) nodes in the network.

Given a large network $G = (V, E)$ (represented by an $n \times n$ adjacency matrix $\mathbf{A}$). we want to find a set of k edges of E such that the removal set creates the largest *drop* of the network robustness according to Equation (1).

Problem 1 MIOBI-BREAKEDGE (Edge Deletion)
Given a large network $G = (V, E)$ (represented by an $n \times n$ adjacency matrix $\mathbf{A}$). With a limited budget k or with k edges removal, we want to find a set of k edges of E such that the removal set creates the largest *drop* of the network robustness according to Equation (1).
More formally, let $\mathbf{A}'$ be an $n \times n$ adjacency matrix of a network $G' = (V, E - S)$ for some set $S \subset E$. We want to find $S \subset E$ of size k such that

$$S \in \arg\max_{S' \subset E : |S'| = k} (\bar{\lambda}^{\mathbf{A}} - \bar{\lambda}^{\mathbf{A}'}).$$

Problem 2 MIOBI-BREAKNODE (Node Deletion)
Given a large network $G = (V, E)$ (represented by an $n \times n$ adjacency matrix $\mathbf{A}$). With a limited budget k or with k nodes removal, we want to find a set of k nodes of V such that the removal set creates the largest *drop* of the network robustness according to Equation (1).
More formally, let $\mathbf{A}'$ be an $n \times n$ adjacency matrix of a network $G' = (V - S, E - T)$ where $T = \{\{u, v\} : u \in S \text{ or } v \in S\}$ for some set $S \subset N$. We want to find $S \subset N$ of size k such that

$$S \in \arg\max_{S' \subset N : |S'| = k} (\bar{\lambda}^{\mathbf{A}} - \bar{\lambda}^{\mathbf{A}'}).$$

4.1.2 Increase the Robustness of the Network

Imagine that you are a system designer of a network and your goal is to increase the fault-tolerance and connectivity of the existing network, or a electricity planner of a power grid network and your goal is to decrease the chance of blackout, or even a cell phone network provider and your goal is to increase the cellular signals or coverage areas. The most natural way to accomplish these tasks is to add new links to the network so that they are more robust then before.

To make the network more robust (or increase the robustness of the network), we consider the network manipulation in which one can add some number of (new) edges to the network.

Problem 3 MIOBI-MAKEEDGE (Edge Addition)
Given a large network $G = (V, E)$ (represented by an $n \times n$ adjacency matrix $\mathbf{A}$). With a limited budget k or with k edges addition, we want to find a set of k edges of E such that the addition set creates the largest *increase* of the network robustness according to Equation (1).

More formally, let $\bar{E} = \{\{u,v\} : u,v \in V \text{ and } \{u,v\} \notin E\}$ be the set of edges not in G (or the complement of E) and let $\mathbf{A}'$ be an $n \times n$ adjacency matrix of a network $G' = (V, E \cup S)$ for some set $S \subset \bar{E}$. We want to find $S \subset \bar{E}$ of size k such that

$$S \in \arg\max_{S' \subset \bar{E} : |S'| = k} (\bar{\lambda}^{\mathbf{A}'} - \bar{\lambda}^{\mathbf{A}}).$$

4.1.3 Other Manipulations

Another way to make the network more robust, is to consider manipulating the network by adding several number of (new) nodes to the network. However, the exact formulation need to be carefully defined. For example, how to set the degree of the nodes to be introduced should be decided.

Another possible graph operation is edge rewiring [3], where existing edges are rewired to connect different pairs of nodes. A principled rewiring can also be done using similar methods to ours; in particular by removing an existing edge via *reverse* MIOBI-BREAKEDGE, and adding it back via MIOBI-MAKEEDGE. Edge rewiring may also enforce that only one end of a to-be-modified edge can be changed.

4.2 Analysis of Problem Hardness

Given the manipulation problems we formulated in the previous sections, the question is "how hard are these problems to solve computationally?" Intuitively, the problems are hard as they involve set selection. In fact, a naive way to solve our manipulation problems is to try all of the possible subsets of size of k of the given set and select the one that will yield the best result. This type of strategy is of course very inefficient. For example, in the case of MIOBI-BREAKEDGE, given a budget k, we have to try all the possible $\binom{m}{k}$ subsets of size k where $|E| = m$. In the case of MIOBI-BREAKNODE, given a budget k, we have to try all the possible $\binom{n}{k}$ subsets of size k where $|V| = n$. In the case of MIOBI-MAKEEDGE, given a budget k, we have to try all the possible $\binom{\binom{n}{2}-m}{k}$ subset of size k.

In this section, we analyze the computational complexity of our problems more formally. We show that the decision versions of MIOBI-BREAKNODE and MIOBI-BREAKEDGE are NP-Complete[1]. We note that it remains a challenge to show the hardness (or the lack thereof) of the MIOBI-MAKEEDGE problem.

4.2.1 MIOBI-BREAKNODE

We show that the optimal k-node deletion problem is NP-Complete. Our solution has two parts. We first show that the decision version of MIOBI-BREAKNODE is in NP and then show that MIOBI-BREAKNODE is NP-Hard.

The decision version of MIOBI-BREAKNODE is stated as:

[1] While we can show the hardness across all the eigenvalues for MIOBI-BREAKNODE and hence for the exact natural connectivity measure as defined in Equ. (1), we are able to do so only for the largest eigenvalue for MIOBI-BREAKEDGE.

P1 (k-node deletion problem MIOBI-BREAKNODE): are there k nodes, the deletion of which makes all the eigenvalues of the graph ≤ 0 and hence natural connectivity $\bar{\lambda} \leq 1$?

Our reduction is from a known NP-Complete problem called the Independent Set (IS) problem:

P2 (k-independent set problem IS): are there k nodes in the graph, no two of which are adjacent?

- **NP:** In order to show that MIOBI-BREAKNODE $\in$ NP, we must show that there exists a polynomial time "witness" algorithm that takes an instance of the problem and a certificate as parameters, and verifies that the certificate is a *yes* instance of the particular input problem. Specifically, our instance is a graph $G(V, E)$, and our certificate is a set of nodes. Our algorithm performs 2 steps: (1) remove all the nodes in the certificate as well as the edges attached to them from the graph; (2) recompute the eigenvalues of the graph and check if all of them are ≤ 0. This algorithm's complexity is dominated by the eigen-decomposition of the adjacency matrix of the graph, which is $O(n^3)$ [33] ($n = |V|$), and is clearly polynomial time. Thus, MIOBI-BREAKNODE $\in$ NP.
- **NP-hard:** To show that MIOBI-BREAKNODE is NP-Hard, we will reduce from IS (as in P2) to our MIOBI-BREAKNODE (as in P1). Our reduction consists of demonstrating a polynomial time conversion of an instance of P2 to an instance of P1, and an if-and-only-if proof that a *yes* instance of P2 maps to a *yes* instance of P1 and vice versa.
  The conversion of an instance of IS to an instance of MIOBI-BREAKNODE works the following way. An instance of IS is a graph G and an integer k. We pass G, and $n - k$ to MIOBI-BREAKNODE, $n = |V|$. Now we will show that a *yes* instance of IS maps to a *yes* instance of MIOBI-BREAKNODE, and vice versa.
  - $\Longrightarrow$ Assume S is a *yes* instance of IS, i.e. there exists an independent set S of size k in G. Thus removing all the rest of the nodes $V \backslash S$ would give us k disconnected nodes with an *all-zero* $k \times k$ adjacency matrix $\bar{A}$. All eigenvalues of a null-matrix is 0 and thus, $\bar{\lambda}(\bar{A}) = \frac{1}{k} \sum_i e^{\lambda_i} = 1$. Thus, the nodes in $V \backslash S$ form a *yes* instance of our MIOBI-BREAKNODE.
  - $\Longleftarrow$ This time assume S is a *yes* instance of MIOBI-BREAKNODE. Thus, after removing S from G, we have $\lambda_i(\tilde{A}) \leq 0$, $\forall i$, where $\tilde{A}$ is the adjacency matrix of the resulting graph. Since $\tilde{A}_{ij} \geq 0, \forall i, j$, i.e. non-negative, by the Perron-Frobenius theory [27] it has a real and non-negative eigenvalue $\lambda_{pf} \geq 0$, where for other eigenvalues λ of $\tilde{A}$, $|\lambda| \leq \lambda_{pf}$. This implies that $\lambda_i(\tilde{A}) = 0$, $\forall i$. A matrix M with all zero eigenvalues is nilpotent, i.e. $M^k = 0$ for some positive integer k. The only non-negative *symmetric integer* nilpotent matrix is the null matrix. Since we also work with undirected (symmetric), un/weighted (binary/integer) graphs, we conclude $\tilde{A}$ to be *all-zero*. As such, nodes in $V \backslash S$ form an independent set of G and thus a *yes* instance of IS.

  Since the conversion of problem instances runs in polynomial time (P2 $\leq_p$ P1) and P2 is NP-Complete, P1 is NP-Hard. As we also show that P1 is in NP, P1 is in fact NP-Complete.

4.2.2 MIOBI-BREAKEDGE

P1 (k-edge deletion problem MIOBI-BREAKEDGE): are there k edges, the deletion of which makes the largest eigenvalue of the graph $\leq \alpha$, for some positive α?

Let us first modify the problem for exact equality,

P1' (modified k-edge deletion problem M-MIOBI-BREAKEDGE): are there k edges, the deletion of which makes the largest eigenvalue of the graph $= \alpha$, for some positive α?

and tie it to the k-clique problem:

P2 (k-clique problem CL): is there a clique of size k in the graph?

- **NP:** It is easy to see that P1' is in NP, since given a graph G we can guess the k edges to be deleted and compute the largest eigenvalue in polynomial time.
- **NP-hard:** In order to show that M-MIOBI-BREAKEDGE $\in$ NP-Hard, we will reduce from CL (as in P2) to our M-MIOBI-BREAKEDGE (as in P1'). The conversion of an instance of CL to an instance of M-MIOBI-BREAKEDGE works the following way. An instance of CL is a graph $G(V,E)$ and an integer k. We pass G, and $e - \frac{(k-1)k}{2}$ and $\alpha = k-1$ to M-MIOBI-BREAKEDGE, $e = |E|$. We show that a *yes* instance of CL maps to a *yes* instance of M-MIOBI-BREAKEDGE, and vice versa.
  - $\Longrightarrow$ Assume C is a *yes* instance of CL, i.e. there exists a clique C of size k in G. Thus removing all the rest of the edges $E \backslash E(C)$ would give us a *all-ones-but-diagonal* $k \times k$ adjacency matrix $\bar{A}$ the eigenvalue of which is k-1. Thus, the edges in $E \backslash E(C)$ form a *yes* instance of our M-MIOBI-BREAKEDGE.
  - $\Longleftarrow$ This time assume S is a *yes* instance of M-MIOBI-BREAKEDGE. Thus, after removing the set S of edges of size $s = e - \frac{(k-1)k}{2}$ from G, we have $\lambda_1(\tilde{A}) = k-1$, where $\tilde{A}$ is the adjacency matrix of the resulting graph $\tilde{G}$. Now we have to show that $\tilde{G}$ can only be a k-clique. A theorem by [32] states that for any graph $G(V,E)$, $\lambda_1(G) \leq \sqrt{2|E|\frac{p-1}{p}}$ where p denotes the size of a maximal clique in the graph and the inequality is sharp if the graph is a clique of size p. From this, we conclude that if $\tilde{G}$ is a k-clique, then $\lambda_1(\tilde{G}) = k-1$. If it is *not* a k-clique, then the maximal clique size $p < k$, and hence $\lambda_1(\tilde{G}) \leq \sqrt{2\frac{(k-1)k}{2}\frac{p-1}{p}} < \sqrt{2\frac{(k-1)k}{2}\frac{k-1}{k}} = k-1$. We thus can conclude that for the largest eigenvalue of $\tilde{G}$ to be as large as $k-1$, it must include a maximal clique of size k. As such, edges in $E \backslash S$ form a k-clique of G and thus a *yes* instance of CL.

  Since the conversion of problem instances runs in poly-time, (P2 $\leq_p$ P1') and P2 is NP-Complete, P1' is NP-Hard. As we also show that P1' is in NP, P1 is in fact NP-Complete.

Next we prove the NP-Completeness of the general edge-deletion problem P1. Our reduction this is time is from the well-known NP-Complete problem Hamiltonian Path:

P3 (Hamiltonian Path problem HP): is there a path that visits every node exactly once in the graph?

- **NP:** It is easy to see that P1 is in NP, as we can guess the k edges to be deleted and compute the largest eigenvalue of the graph in poly-time.
- **NP-hard:** In order to show that MIoBI-BREAKEDGE $\in$ NP-Hard, we will reduce from HP (as in P3) to our MIoBI-BREAKEDGE (as in P1). The conversion of an instance of HP to an instance of MIoBI-BREAKEDGE works the following way. An instance of HP is a graph $G(V,E)$ with $|E| = e$ edges. We pass G, and $k = e - (n-1)$ and $\alpha = 2\cos(\frac{\pi}{n+1})$ to MIoBI-BREAKEDGE, for $|V| = n$. We show that a *yes* instance of HP maps to a *yes* instance of MIoBI-BREAKEDGE, and vice versa.
  - $\Longrightarrow$ Assume P is a *yes* instance of HP, i.e. there exists a path P of length $n-1$ in G. Then removing all the rest of the edges $E \backslash E(P)$ would give us a chain graph the principal eigenvalue of which is $2\cos(\frac{\pi}{n+1})$. Thus, the edges in $E \backslash E(P)$ form a *yes* instance of our MIoBI-BREAKEDGE.
  - $\Longleftarrow$ This time assume S is a *yes* instance of MIoBI-BREAKEDGE. Thus, after removing the set S of edges of size $|S| = e - (n-1)$ from G, we have $\lambda_1(\tilde{A}) \leq 2\cos(\frac{\pi}{n+1})$, where $\tilde{A}$ is the adjacency matrix of the resulting graph $\tilde{G}$. Now we have to show that $\tilde{G}$ can only be a Hamiltonian Path. Assume that $\tilde{G}$ is *not* a Hamiltonian Path. If it is not a path but connected, then it must be a tree. According to a theorem by [44], a path P_n visiting n nodes has strictly smaller spectral radius (largest eigenvalue) than all other connected graphs with n nodes, where $\lambda_1(P_n) = 2\cos(\frac{\pi}{n+1})$. This ensures that $\tilde{G}$ is a path, if it is connected. Next we show that it cannot be disconnected. Without loss of generality, assume $\tilde{G}$ has two components, one with x nodes and another with $n-x$ nodes. Since those are connected, they must contain at least $x-1$ and $n-x-1$ edges respectively. Since there exists $n-1$ edges in total, either one of the components should contain the extra edge, creating a cycle in that component. A graph that contains a cycle has largest eigenvalue ≥ 2, which contradicts P1 with $\alpha \leq 2\cos(\frac{\pi}{n+1}) < 2$. Therefore, we conclude that $\tilde{G}$ should be a connected graph, and that $\tilde{G}$ is a (connected) path. A path on all n nodes of a graph is a Hamiltonian Path. As such, edges in $E \backslash S$ form a *yes* instance of HP.

  Since the conversion of problem instances runs in poly-time, (P3 $\leq_p$ P1) and P3 is NP-Complete, P1 is NP-Hard. Having shown that P1 is also in NP, we conclude that P1 is NP-Complete.

4.3 Our Approach

Our approach to tackle these problems is first compute values to quantify edges/nodes based on the natural connectivity scores and then select edges/nodes based on the measurements. There are two basic ways in which edges/nodes can be scored and selected: *Greedy* and *Adaptive*.

In the case of *Greedy*, the values for edges/nodes are computed exactly once (for the unmodified original network) for a specific method. Then we rank the edges/nodes accordingly based on the computed values and select the top k candidates to remove/add one step at a time.

In the case of *Adaptive*, the values for edges/nodes are recomputed each time after each operation (and hence the iterative update of our eigenvalues/eigenvectors). At each step, we rank the edges/nodes accordingly base on the computed values and select the top candidate to remove/add. We then repeat this process k times/operations.

This cautious edge deletion strategy (or in our context *Adaptive*), has been used in prior research [19,8] where for example edge betweenness is used as criterion to remove edges to decrease the robustness the most. In their papers, instead of choosing the top-k edges with the highest edge betweenness in one shot, their idea is to remove a single edge in each step after which the betweenness is updated for the remaining, where this procedure is repeated for the next k steps. This is often referred as the "recalculated" (or "iteratively-selected") strategy and has been shown to perform better (i.e., affect robustness more) compared to its top-k counterpart.

Therefore, to find the k most effective edges, we will follow a more unorthodox yet cautious strategy (Adaptive) which will iteratively find the best single edge to remove, for k steps. That is, every time an edge is removed the criterion/score that is used to find an edge to remove will be updated for the remaining edges. That is because each removed edge changes this score as it changes the robustness.

Clearly, Adaptive is more expensive than Greedy in computation time. Moreover, one would expect that Adaptive will be more effective than Greedy using our methods. In the experiment, we will show that indeed Adaptive is more effective than Greedy empirically on 16 real-world datasets.

In the following subsections, we introduce our methods to solve each of the problems efficiently. In particular, the running time and space of our algorithms for MIoBI-BreakEdge and MIoBI-BreakNode are linear in the size of the graph. The running time of our algorithm for MIoBI-MakeEdge is sub-quadratic and takes linear space.

4.3.1 "Breaking" Network Robustness

Next we describe our proposed algorithms to solve the problems formulated in the previous section. Note that the first two problems aim at maximally decreasing (i.e., "breaking"), whereas the third problem aims at improving (i.e., "making") network robustness.

Solution to Problem 1: Edge Deletion

First, we address the edge deletion problem MIoBI-BreakEdge, which aims to find the set of k edges to delete from the graph so that the robustness is shrunk the most.

Let S denote the selected set of k edges to be removed. Let us then write the new, updated robustness $\bar{\lambda}_{\Delta}$ as

$$\bar{\lambda}_{\Delta} = \ln(\frac{1}{n}\sum_{j=1}^{n} e^{\lambda_j + \Delta\lambda_j}) \tag{2}$$

where $\Delta\lambda_j$ is the difference in eigenvalue λ_j after the adjustment to the graph. Therefore, we need to be able to efficiently update the eigenvalues of $\mathbf{A}$ when the graph changes in order to quantify the updated robustness of the graph in the face of change.

Updating the eigenvalues. Using the first order matrix perturbation theory [41], we can compute changes to the eigenvalues $\Delta\lambda_j$ efficiently.

Let $(\lambda_j, \mathbf{u_j})$ be the j^{th} (eigenvalue, eigenvector) pair of the graph G with adjacency matrix $\mathbf{A}$. Let $\Delta\mathbf{A}$ and $(\Delta\lambda_j, \Delta\mathbf{u_j})$ denote the change in $\mathbf{A}$ and $(\lambda_j, \mathbf{u_j})$ $\forall j$, respectively (where $\Delta\mathbf{A}$ is symmetric). Suppose after the adjustment $\mathbf{A}$ becomes

$$\tilde{\mathbf{A}} = \mathbf{A} + \Delta\mathbf{A}$$

where $(\tilde{\lambda_j}, \tilde{u_j})$ is written as

$$\tilde{\lambda_j} = \lambda_j + \Delta\lambda_j$$
$$\tilde{\mathbf{u_j}} = \mathbf{u_j} + \Delta\mathbf{u_j}$$

Lemma 1 *Given a perturbation $\Delta\mathbf{A}$ to a matrix $\mathbf{A}$, its eigenvalues can be updated by*

$$\Delta\lambda_j = \mathbf{u_j}'\Delta\mathbf{A}\mathbf{u_j}. \tag{3}$$

PROOF of *Lemma 1*. We can write

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

Expanding the above, we get

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

By concentrating on first-order approximation, we assume that all high-order perturbation terms are negligible, including $\Delta\mathbf{A}\Delta\mathbf{u_j}$ and $\Delta\lambda_j\Delta\mathbf{u_j}$. Further, by using the fact that $\mathbf{A}\mathbf{u_j} = \lambda_j\mathbf{u_j}$ (i.e., canceling these terms) we obtain

$$\Delta\mathbf{A}\mathbf{u_j} + \mathbf{A}\Delta\mathbf{u_j} = \Delta\lambda_j\mathbf{u_j} + \lambda_j\Delta\mathbf{u_j} \tag{4}$$

Next we multiply both sides by $\mathbf{u_j}'$ and by symmetry of $\mathbf{A}$ and orthonormal property of its eigenvectors we get Equ. (3), which concludes the proof.

□

Using *Lemma 1*, perturbing $\mathbf{A}$ with any given edge (p, r) affects the eigenvalues as

$$\Delta\lambda_j = \mathbf{u_j}'\Delta\mathbf{A}\mathbf{u_j} = -2\mathbf{u_{pj}}\mathbf{u_{rj}} \tag{5}$$

where $\Delta\mathbf{A}(p, r) = \Delta\mathbf{A}(r, p) = -1$ and 0 elsewhere.

As such, for *Problem 1* we are interested in k edges that will minimize $\bar{\lambda}_\Delta$ in Equ. (2), or equivalently

$$\begin{aligned} \min \quad & e^{\lambda_1+\Delta\lambda_1} + e^{\lambda_2+\Delta\lambda_2} + \ldots + e^{\lambda_n+\Delta\lambda_n} \\ & e^{\lambda_1}(e^{\Delta\lambda_1} + e^{(\lambda_2-\lambda_1)}e^{\Delta\lambda_2} + \ldots + e^{(\lambda_n-\lambda_1)}e^{\Delta\lambda_n}) \\ & c_1(e^{\Delta\lambda_1} + c_2e^{\Delta\lambda_2} + \ldots + c_ne^{\Delta\lambda_n}) \end{aligned} \tag{6}$$

where c_j's denote constant terms and $c_j \le 1, \forall j \ge 2$.

Therefore, following the re-calculated strategy and by using Equ. (5) and (6) we will choose the edge (p,r) that minimizes the following:

$$\min_{(p,r)\in E} \quad c_1\left(e^{-2\mathbf{u_{p1}}\mathbf{u_{r1}}} + c_2 e^{-2\mathbf{u_{p2}}\mathbf{u_{r2}}} + \ldots + c_n e^{-2\mathbf{u_{pn}}\mathbf{u_{rn}}}\right) \tag{7}$$

Our criterion/score to select edges to remove as given in Equ. (7) changes whenever an edge is removed, as the graph structure and thus eigenvectors $\mathbf{u_j}$ change. Thus, after every step we also need to update the eigenvectors. The key question is how to compute changes $\Delta\mathbf{u_j}$ efficiently. For that, we again resort to matrix perturbation theory [41].

Updating the eigenvectors.

Lemma 2 *Given a perturbation $\Delta\mathbf{A}$ to a matrix $\mathbf{A}$, its eigenvectors can be updated by*

$$\Delta\mathbf{u_j} = \sum_{i=1,i\neq j}^{n}\left(\frac{\mathbf{u_i}'\Delta\mathbf{A}\mathbf{u_j}}{\lambda_j-\lambda_i}\mathbf{u_i}\right). \tag{8}$$

PROOF of *Lemma 2*. Using the orthogonality property of the eigenvectors, we can write the change $\Delta\mathbf{u_j}$ of eigenvector $\mathbf{u_j}$ as a linear combination of the original eigenvectors:

$$\Delta\mathbf{u_j} = \sum_{i=1}^{n}\alpha_{ij}\mathbf{u_i} \tag{9}$$

where α_{ij}'s are small constants that we aim to determine.

Using Equ. (9) in Equ. (4) from the proof of *Lemma 1*, we obtain

$$\Delta\mathbf{A}\mathbf{u_j} + \mathbf{A}\sum_{i=1}^{n}\alpha_{ij}\mathbf{u_i} = \Delta\lambda_j\mathbf{u_j} + \lambda_j\sum_{i=1}^{n}\alpha_{ij}\mathbf{u_i}$$

which is equivalent to

$$\Delta\mathbf{A}\mathbf{u_j} + \sum_{i=1}^{n}\lambda_i\alpha_{ij}\mathbf{u_i} = \Delta\lambda_j\mathbf{u_j} + \lambda_j\sum_{i=1}^{n}\alpha_{ij}\mathbf{u_i}$$

Multiplying both sides of the above equation by $\mathbf{u_k}'$, $k\neq j$, we get

$$\mathbf{u_k}'\Delta\mathbf{A}\mathbf{u_j} + \lambda_k\alpha_{kj} = \lambda_j\alpha_{kj}$$

Therefore,

$$\alpha_{kj} = \frac{\mathbf{u_k}'\Delta\mathbf{A}\mathbf{u_j}}{\lambda_j-\lambda_k} \tag{10}$$

for $k\neq j$. To obtain α_{jj} we use the following derivation.

$$\begin{aligned}\tilde{\mathbf{u_j}}'\tilde{\mathbf{u_j}} = 1 &\Rightarrow (\mathbf{u_j}+\Delta\mathbf{u_j})'(\mathbf{u_j}+\Delta\mathbf{u_j}) = 1\\ &\Rightarrow 1 + 2\mathbf{u_j}'\Delta\mathbf{u_j} + \|\Delta\mathbf{u_j}\|^2 = 1\end{aligned}$$

After we discard the high-order term, and substitute $\Delta\mathbf{u_j}$ with Equ. (9) we get $1 + 2\alpha_{jj} = 1 \Rightarrow \alpha_{jj} = 0$.

We note that for a slightly better approximation, one can choose not to ignore the high-order term which is equal to $\|\Delta\mathbf{u_j}\|^2 = \sum_{i=1}^{n} \alpha_{ij}^2$. Thus, one can compute α_{jj} as

$$1 + 2\alpha_{jj} + \sum_{i=1}^{n} \alpha_{ij}^2 = 1 \Rightarrow 1 + 2\alpha_{jj} + \alpha_{jj}^2 + \sum_{\substack{i=1 \\ i\neq j}}^{n} \alpha_{ij}^2 = 1$$

$$\Rightarrow (1+\alpha_{jj})^2 + \sum_{\substack{i=1 \\ i\neq j}}^{n} \alpha_{ij}^2 = 1 \Rightarrow \alpha_{jj} = \sqrt{1 - \sum_{\substack{i=1 \\ i\neq j}}^{n} \alpha_{ij}^2} - 1$$

All in all, using the α_{ij}'s as given by Equ. (10) and $\alpha_{jj} = 0$, we can see that $\Delta\mathbf{u_j}$ in Equ. (9) is equal to Equ. (8).

□

Finally, we remark that it is infeasible to compute all the n eigenvalues of graphs with n nodes, for very large n. Luckily, given the skewed spectrum of real-world graphs [14], only the top few eigenvalues have large magnitudes which implies that the c_j terms in Equ.s (6,7) become smaller and smaller for increasing j. Therefore, we will focus on and compute the top t eigenvalues to approximate the robustness of a graph in our experiments.

The detailed pseudo-code of our algorithm for the edge deletion problem MIOBI-BREAKEDGE is given as follows.

Algorithm 1 MIOBI-BREAKEDGE

```
Input: Graph $G(V,E)$, its adj. matrix $\mathbf{A}$, and int. budget $k$
Output: Set $S$ of $k$ edges to be removed
 1: $S = \emptyset$
 2: Compute the top $t$ (eigenvalue, eigenvector) pairs $(\lambda_j, \mathbf{u_j})$ of $\mathbf{A}$, $1 \leq j \leq t$
 3: for $step = 1$ to $k$ do
 4:     Select the edge $(\bar{p},\bar{r})$ out of $\forall(p,r) \in E$ that minimizes Equ. (7) for top $t$ eigenvectors, i.e.
            $\min_{(p,r)\in E} c_1\left(e^{-2\mathbf{u_{p1}u_{r1}}} + c_2 e^{-2\mathbf{u_{p2}u_{r2}}} + \ldots + c_t e^{-2\mathbf{u_{pt}u_{rt}}}\right)$
        where $c_1 = e^{\lambda_1}$ and $c_j = e^{(\lambda_j - \lambda_1)}$ for $2 \leq j \leq t$
 5:     $S := S \cup (\bar{p},\bar{r})$, $E := E \backslash (\bar{p},\bar{r})$
 6:     Update $\mathbf{A}$; $\mathbf{A}(\bar{p},\bar{r}) = 0$ and $\mathbf{A}(\bar{r},\bar{p}) = 0$
 7:     Update top $t$ eigenvalues of $\mathbf{A}$ by Equ. (3)
 8:     Update top $t$ eigenvectors of $\mathbf{A}$ by Equ. (8)
 9: end for
10: Return $S$
```

Complexity Analysis. The efficiency of the proposed Algorithm 1 is given in the following lemma; for a fixed budget k, MIOBI-BREAKEDGE is linear with respect to the size of the graph for both time and space cost.

Lemma 3 Complexity of MIoBI-BreakEdge. *The time cost of Alg. 1 is $O(kmt + knt^2)$. The space cost of Alg. 1 is $O(m + nt + k)$.*

PROOF SKETCH. Computing top t eigenvalues and eigenvectors (step 2) takes $O(nt + mt + nt^2)$ using iterative approximate methods by Lanczos [23]. Computing scores (step 4) takes $O(mt)$. Updating eigenvalues (step 7) and eigenvectors (step 8) takes $O(t)$ and $O(nt^2)$, respectively. Overall complexity for k iterations is then $O(k(mt + nt^2))$.

Storing the edge-list of graph G requires $O(m)$. The eigenvalue, eigenvector pairs take $O(t)$ and $O(nt)$ respectively. We find the edge with minimum score in each iteration with $O(1)$ space. Finally, $O(k)$ is required to store selected edges. Overall, space cost is $O(m + nt + k)$. □

Solution to Problem 2: Node Deletion

Next, we address the node deletion problem MIoBI-BreakNode, which aims to find the set S of k nodes to delete from the graph so that the robustness is reduced the most. Deletion of a node involves deletion of the node as well as all its incident edges, i.e. edges attached to it.

Similar to edge deletion, we can delete nodes from the graph one by one (i.e., recalculated strategy). To do so, we need to find a score similar to Equ. (7) for each node to quantify its effect of change on the graph spectrum. Again, using $\Delta\lambda_j = \mathbf{u_j}'\Delta\mathbf{A}\mathbf{u_j}$ from *Lemma 1*, we will write down a score for each node i where only the i^{th} row and i^{th} column of $\Delta\mathbf{A}$ contain non-zero entries; $(i,v) = (v,i) = -1$, $v \in \mathcal{N}(i)$, for neighbors $\mathcal{N}(i)$ of i.

We can illustrate the node scoring with a toy example, where say we are to remove a node i with 3 neighbors indexed by n_1, n_2, n_3. Let $\mathbf{w_j} = \mathbf{u_j}'\Delta\mathbf{A}$. We can see that $\mathbf{w_{n_1j}} = \mathbf{w_{n_2j}} = \mathbf{w_{n_3j}} = -\mathbf{u_{ij}}$, and $\mathbf{w_{ij}} = -\sum_{v\in N(i)} \mathbf{u_{vj}}$. As such, $\Delta\lambda_j = \mathbf{w_j}\mathbf{u_j} = -\mathbf{u_{ij}}\mathbf{u_{n_1j}} - \mathbf{u_{ij}}\mathbf{u_{n_2j}} - \mathbf{u_{ij}}\mathbf{u_{n_3j}} - \sum_{v\in\mathcal{N}(i)} \mathbf{u_{vj}}\mathbf{u_{ij}}$, equivalently $\Delta\lambda_j = -\mathbf{u_{ij}}(\mathbf{u_{n_1j}} + \mathbf{u_{n_2j}} + \mathbf{u_{n_3j}} + \sum_{v\in\mathcal{N}(i)} \mathbf{u_{vj}}) = -2\mathbf{u_{ij}} \sum_{v\in\mathcal{N}(i)} \mathbf{u_{vj}}$.

Thus, in general $\Delta\lambda_j$ for a removal of node i is given as

$$\Delta\lambda_j = \mathbf{u_j}'\Delta\mathbf{A}\mathbf{u_j} = -2\mathbf{u_{ij}} \sum_{v\in\mathcal{N}(i)} \mathbf{u_{vj}} \tag{11}$$

Equ. (11) essentially states that the change in the j^{th} eigenvalue for a node i's removal is twice as the sum of eigenscores of i's neighbors multiplied by the eigenscore of i, where eigenscores denote the corresponding entries in the associated j^{th} eigenvector.

For *Problem 2* we are interested in selecting k nodes that will minimize $\bar{\lambda}_\Delta$ in Equ. (2). As we will select the nodes iteratively one by one, we will pick the node i that minimizes the following at every step.

$$\min_{i\in V} \quad c_1\left(e^{-2\mathbf{u_{i1}} \sum\limits_{v\in N(i)} \mathbf{u_{v1}}} + \ldots + c_n e^{-2\mathbf{u_{in}} \sum\limits_{v\in\mathcal{N}(i)} \mathbf{u_{vn}}}\right) \tag{12}$$

where c_j's denote the constants as before. Note that we will also consider only the top t eigenvectors to compute the node selection scores in the experiments.

Algorithm 2 for the node deletion problem MIOBI-BREAKNODE follows similar lines as of the algorithm for MIOBI-BREAKEDGE, where we use Equ. (12) instead of Equ. (7) in Line 4 of Algorithm 1.

Algorithm 2 MIOBI-BREAKNODE

```
Input: Graph $G(V, E)$, its adj. matrix $\mathbf{A}$, and int. budget $k$
Output: Set $S$ of $k$ nodes to be removed
 1: $S = \emptyset$
 2: Compute the top $t$ (eigenvalue, eigenvector) pairs $(\lambda_j, \mathbf{u_j})$ of $\mathbf{A}$, $1 \leq j \leq t$
 3: for $step = 1$ to $k$ do
 4:     Select the node $\bar{i}$ out of $\forall i \in V$ that minimizes Equ. (12) for top $t$ eigenvectors, i.e.
            $$\min_{i \in V} \quad c_1 \left( e^{-2\mathbf{u_{i1}} \sum_{v \in N(i)} \mathbf{u_{v1}}} + \ldots + c_t e^{-2\mathbf{u_{it}} \sum_{v \in \mathcal{N}(i)} \mathbf{u_{vt}}} \right)$$
        where $c_1 = e^{\lambda_1}$ and $c_j = e^{(\lambda_j - \lambda_1)}$ for $2 \leq j \leq t$
 5:     $S := S \cup \bar{i}, V := V \backslash \bar{i}, E := E \backslash (\bar{i}, v),\ v \in \mathcal{N}(\bar{i})$
 6:     Update $\mathbf{A}$; $\mathbf{A}(:, \bar{i}) = 0$ and $\mathbf{A}(\bar{i}, :) = 0$
 7:     Update top $t$ eigenvalues of $\mathbf{A}$ by Equ. (3)
 8:     Update top $t$ eigenvectors of $\mathbf{A}$ by Equ. (8)
 9: end for
10: Return $S$
```

Complexity Analysis. MIOBI-BREAKNODE is also linear in graph size for both time and space cost.

Lemma 4 Complexity of MIOBI-BREAKNODE. *The time cost of Alg. 2 is $O(kmt + knt^2)$. The space cost is $O(m + nt + k)$.*

PROOF SKETCH. Computing top t eigenvalues and eigenvectors: $O(nt + mt + nt^2)$. Computing scores: $O(mt)$. Updating eigenvalues/vectors: $O(t)$ and $O(nt^2)$. Overall for k iterations: $O(k(mt + nt^2))$.

Storing the graph: $O(m)$. The eigenvalue/vector pairs: $O(t)$ and $O(nt)$. Min-scoring node in each iteration: $O(1)$. Selected nodes: $O(k)$. Overall: $O(m + nt + k)$. $\square$

4.3.2 "Making" Network Robustness

Solution to Problem 3: Edge Addition

In this section, we address the edge addition problem MIOBI-MAKEEDGE, to find the set of k edges to place to the graph so that the robustness is improved the most.

MIOBI-MAKEEDGE is a harder and computationally more demanding problem than MIOBI-BREAKEDGE, since there are $O(n^2)$ potential edges to add to a given graph (compared to $O(m)$ edges to remove). As for large graphs quadratic operations are not desirable, we need to design an algorithm that is fast and that scales well.

Similar to deletion, we will adopt the re-calculated strategy for edge additions and find the k edges one by one iteratively. As such, at every iteration, we are interested in finding the edge that maximizes the following.

$$\max_{\substack{(p,r)\notin E\\ p\in V, r\in V}} \quad c_1\left(e^{2\mathbf{u_{p1}}\mathbf{u_{r1}}} + c_2 e^{2\mathbf{u_{p2}}\mathbf{u_{r2}}} + \ldots + c_n e^{2\mathbf{u_{pn}}\mathbf{u_{rn}}}\right) \tag{13}$$

According to the Perron-Frobenius theorem [17], the principal eigenvector associated with the largest eigenvalue of non-negative irreducible matrices has all positive entries. As $G(V, E)$ is a connected undirected graph, $\mathbf{A}$ is irreducible and $\mathbf{u_1}$ is a positive vector. On the other hand other $\mathbf{u_j}$'s, $j > 1$, might potentially have negative entries. This makes finding the edge that maximizes Equ. (13) without enlisting all $O(n^2)$ edges challenging.

Next we introduce a fast approximation strategy to pick edges to add without enlisting all possible edges. In particular, we note that the second and onwards terms in Equ. (13) keep getting smaller and smaller, due to the skewed spectrum of large real-world graphs [43,26]. Therefore, we focus on the first term, i.e. $e^{2\mathbf{u_{p1}}\mathbf{u_{r1}}}$. We create a set $\mathcal{C} \subset V$ of size $d_{\max}$, where $d_{\max}$ denotes the maximum node degree in G, that consists of the nodes with highest $\mathbf{u_1}$ entries. For all non-edges (p, r) of G, $p \in \mathcal{C},\ r \in \mathcal{C},\ p \neq r$, we compute Equ. (13) considering the top t eigenvectors, and we add the edge $(\bar{p}, \bar{r})$ with the maximum value. We repeat this procedure k times. Algorithm 3 gives the steps of our proposed edge addition algorithm in detail.

Algorithm 3 MIOBI-MAKEEDGE

```
Input: Graph $G(V,E)$, its adj. matrix $\mathbf{A}$, and int. budget $k$
Output: Set $S$ of $k$ edges to be added
 1: $S = \emptyset$
 2: Compute the top $t$ (eigenvalue, eigenvector) pairs $(\lambda_j, \mathbf{u_j})$ of $\mathbf{A}$, $1 \le j \le t$
 3: for $step = 1$ to $k$ do
 4:     Compute the largest degree $d_{\max}$ of $\mathbf{A}$
 5:     Find the candidate subset $\mathcal{C}$ of $d_{\max}$ nodes with the highest $\mathbf{u_1}$ eigen-scores
 6:     Select the edge $(\bar{p},\bar{r})$ out of $\forall(p,r) \notin E, p \in \mathcal{C},\ r \in \mathcal{C},\ p \neq r$, that maximizes Equ. (13) for top
        $t$ eigenvectors, i.e.
        $$\max_{\substack{(p,r)\notin E\\ p\in\mathcal{C}, r\in\mathcal{C}}} \quad c_1\left(e^{2\mathbf{u_{p1}}\mathbf{u_{r1}}} + c_2 e^{2\mathbf{u_{p2}}\mathbf{u_{r2}}} + \ldots + c_t e^{2\mathbf{u_{pt}}\mathbf{u_{rt}}}\right)$$
 7:     $S := S \cup (\bar{p},\bar{r})$, $E := E \cup (\bar{p},\bar{r})$
 8:     Update $\mathbf{A}$; $\mathbf{A}(\bar{p},\bar{r}) = 1$ and $\mathbf{A}(\bar{r},\bar{p}) = 1$
 9:     Update top $t$ eigenvalues of $\mathbf{A}$ by Equ. (3)
10:     Update top $t$ eigenvectors of $\mathbf{A}$ by Equ. (8)
11: end for
12: Return $S$
```

Complexity Analysis. MIOBI-MAKEEDGE is linear in graph size for space, and sub-quadratic for time cost.

Lemma 5 Complexity of MIoBI-MakeEdge. *The time cost of Alg. 3 is $O(mt + kd_{\max}^2 t + knt^2)$. The space cost is $O(m + nt + k)$.*

PROOF SKETCH. Computing top t eigenvalues and eigenvectors: $O(nt + mt + nt^2)$. Computing scores: $O(d_{\max}^2 t)$. Updating eigenvalues/vectors: $O(t)$ and $O(nt^2)$. Overall for k iterations: $O(mt + k(d_{\max}^2 t + nt^2))$.

Storing the graph: $O(m)$. The eigenvalues/vectors: $O(t)$ and $O(nt)$. Max-scoring edge in each iteration: $O(1)$. Selected edges: $O(k)$. Overall: $O(m + nt + k)$. □

4.3.3 Higher Order Approximation for Eigenvectors

Before we conclude this section, we discuss a potentially 'better' method to update the eigenvectors (we note that the estimation of $\varDelta\lambda_j (j = 1, ..., t)$ remains the same as before, i.e., by using Equ. (3)).

In *Lemma 2*, we establish that we can update $\varDelta\mathbf{u_j}$ via Equ. (8) quite efficiently. In the proof of *Lemma 2*, we write the change $\varDelta\mathbf{u_j}$ as a linear combination of the original eigenvectors:

$$\varDelta\mathbf{u_j} = \sum_{i=1}^{t} \alpha_{ij}\mathbf{u_i} \tag{14}$$

then substitute the expression into Equ. (4) and solve for the α_{ij} terms.

However, Equ. (4) is obtained from Equ. (4) by removing high-order perturbation terms. It appears that if we plug Equ. (14) into Equ. (4) and keep the higher order terms, we could provide a better estimation of the eigenvectors by selecting 'better' α values.

Now, suppose that we plug Equ. (14) into Equ. (4) and multiply both sides by $\mathbf{u}_k'$ $(k = 1, ..., t)$, $k \neq j$. Then, for a given $\mathbf{u}_j$ $(j = 1, ..., t)$ we have

$$\mathbf{X}(k, j) + (\lambda_k - \lambda_j - \varDelta\lambda_j)\alpha_{k,j} + \sum_{i=1}^{t} \mathbf{X}(k, i)\alpha_{i,j} = 0 \tag{15}$$

where $\mathbf{X}(k, i) = \mathbf{u}_k'\varDelta\mathbf{A}\mathbf{u}_i$ $(k, i = 1, ..., t)$.

For simplicity, we let $\alpha_j = [\alpha_{1,j}, ..., \alpha_{k,j}]'$ and $\mathbf{D} = \text{diag}(\lambda_j + \varDelta\lambda_j - \lambda_k)$ for $k = 1, ..., t$. In this matrix form, we have the following linear system for α_j:

$$(\mathbf{D} - \mathbf{X})\alpha_j = \mathbf{X}(:, j)$$

Solving for α_j, we have

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

We can see that the above new formula includes the original estimation in Equ. (10) as a special case by dropping $\varDelta\lambda_j$ and $\mathbf{X}$. Theoretically, this will give us a better estimation, and help us get around the multiplicity issue of the eigenvalues. However, this method of update is more expensive; the new formula involves a matrix inverse and increases the time complexity by an additional t^4 where t is the number of eigenvectors to be updated. Thus, it appears that Equ. (8) is more appropriate for large networks, which we will use to update the eigenvectors in this paper.

5 Experimental Evaluation

We evaluate our algorithms with respect to (A) effectiveness in manipulating graph robustness, and (B) running time and scalability, on several real-world graphs. For each kind of graph manipulation, i.e. problem setting, we compare to several ad-hoc heuristic strategies that we compiled.

Datasets. We use the datasets shown in Table 2 (all available at `http://snap.stanford.edu/data/`) to evaluate our methods.

– **Oregon Autonomous System (AS) Router Network** This network is AS-level connectivity networks inferred from Oregon route-views, and were collected once a week, for 9 consecutive weeks. The nine Oregon datasets, which we name as *Oregon-A* through *Oregon-I*, correspond to each of the weeks.

– **Gnutella Peer-to-Peer Network** The Gnutella graph is the peer-to-peer (P2P) connectivity networks collected daily, for 5 consecutive days. The five Gnutella datasets, which we name as *P2P-GnutellaA* through *P2P-GnutellaE*, correspond to each of the days.

– **Wikipedia Vote Network** When a user of Wikipedia submit a request to become an administrator, an election will be hold publicly and any user of Wikipedia can participate or vote in the election. The Wikipedia Vote (Wiki-Vote) network dataset contains such election data. Wiki-Vote contains 2,794+ elections with 103,689 total number of votes and 7,115 voters and candidates. Naturally, an edge in this graph corresponds to a user voted for the candidate in an election.

– **Enron Email Network** This dataset is resultant from an investigation by the Federal Energy Regulatory Commission (FERC) to track email communication between different email addresses. The edges in the network correspond to at least once email between the email addresses (nodes of the network). There are 36,692 nodes (email addresses) and 367,662 edges in this network.

Evaluation criteria. Recall in this paper, we will use the natural connectivity measure (as shown in an earlier section) to quantify the robustness of the network $G(V, E)$

$$\bar{\lambda} = \ln\left(\frac{1}{n}\sum_{j=1}^{n} e^{\lambda_j}\right) \tag{16}$$

which corresponds to an "average" eigenvalue of G, where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ denote a non-increasing ordering of the eigenvalues of its adjacency matrix $\mathbf{A}$. For ease of computation, to compute the robustness of the network using the natural connectivity measure, we only compute and use the first 25 eigenvalues (i.e. $n = 25$) to approximate $\bar{\lambda}$. Hence, throughout this experimental section, we report the robustness of the network $G(V, E)$ based on the first 25 eigenvalues.

To measure the effectiveness of our methods, we report the relative $\%$ change of robustness

$$\%\Delta R \equiv \frac{|R_{initial} - R_{final}|}{R_{initial}} \times 100\%$$

Table 2: Dataset summary.

Dataset	n	m	$density$
Oregon-A	633	1,086	0.0054
Oregon-B	1,503	2,810	0.0024
Oregon-C	2,504	4,723	0.0015
Oregon-D	2,854	4,932	0.0012
Oregon-E	3,995	7,710	0.0009
Oregon-F	5,296	10,097	0.0007
Oregon-G	7,352	15,665	0.0005
Oregon-H	10,860	23,409	0.0004
Oregon-I	13,947	30,584	0.0003
P2P-GnutellaA	6,301	20,777	0.0010
P2P-GnutellaB	8,114	26,013	0.0008
P2P-GnutellaC	8,717	31,525	0.0008
P2P-GnutellaD	8,846	31,839	0.0008
P2P-GnutellaE	10,876	39,994	0.0007
Wiki-Vote	7,115	100,762	0.0040
Email-Enron	36,692	183,811	0.0003

where $R_{initial}$ and R_{final} denote the initial and the final robustness after k operations, respectively. The initial robustness score $R_{initial}$ is computed using the original network without any modification. After performing k operations, we recompute the final robustness score R_{final} of the same network. Clearly, the larger of $\%$ change in robustness, the more effective in manipulating the robustness of the network.

To measure computational cost and show that our methods are feasible for large networks, we report the wall-clock time in seconds. The wall-clock time is simply the time will take for our methods to complete the k operations. All the reported times are on a 64-bit machine using Intel Core i5-3570K CPU @3.40GHz and 8GB memory, running Ubuntu 12.10 (Kernel: Linux ubuntu 3.5.0-17-generic x86_64).

Set up. Recall that in our methods, we need to determine the number of eigen-pairs to be used in the approximation. For the experiments here, we used top $t = 50$ eigen-pairs. Due to the iterative-update nature of our methods for eigen-pairs, for large perturbations to the graph (e.g., high degree nodes removed for node deletions), the accumulated error for updating eigenvalues and vectors (using Equations (3)&(8)) increases rapidly and the performance degrades. To overcome this issue, we recompute the exact eigen-pairs of the perturbed graph at every 50 operations. We call our former, always-updated methods 'Naive' and the recomputed ones as 'RC@50'.

Greedy vs. Adaptive. As mentioned in the earlier section, Greedy and Adaptive are the two ways that edges/node can be scored and selected. We claim that Adaptive will be more effective than the Greedy using our methods and we will show this is true, empirically, in this section. To show this, we compute the percentage of relative robustness improvement from Greedy to Adaptive

$$\%R_{improve}^{Greedy \rightarrow Adaptive} = \frac{\%\Delta R_{Adaptive} - \%\Delta R_{Greedy}}{\%\Delta R_{Greedy}} \times 100\%$$

Table 3: MIOBI-BREAKEDGE: % of relative robustness improvement from Greedy to Adaptive

Dataset	k=1%*m	k=5%*m	k=10%*m	k=15%*m	k=25%*m
O-A	0.00	5.89	3.51	3.22	8.39
O-B	1.13	6.29	8.20	10.96	16.13
O-C	4.60	5.95	12.26	13.76	18.56
O-D	4.09	7.56	14.20	30.96	8.87
O-E	4.84	4.70	14.36	21.55	15.76
O-F	5.78	5.87	14.36	20.55	19.14
O-G	9.19	21.63	13.90	19.74	15.15
O-H	10.15	24.85	15.33	19.40	14.08
O-I	12.96	32.32	18.58	23.91	20.12
P2P-A	0.67	18.42	20.68	7.15	11.38
P2P-B	4.16	23.71	11.11	7.93	12.35
P2P-C	41.69	61.15	16.85	17.81	19.67
P2P-D	37.30	20.10	10.30	15.96	17.82
P2P-E	46.66	20.80	28.60	31.03	32.48
W-V	1.34	3.88	6.39	8.33	13.26
E-E	2.18	21.87	49.15	55.43	37.76

where $\%\Delta R_{Adaptive}$ and $\%\Delta R_{Greedy}$ are the relative $\%$ change of robustness of our methods under Adaptive and Greedy, respectively. Tables 3, 4, and 5 show the percentage of relative robustness improvement of MIOBI-BREAKEDGE, MIOBI-BREAKNODE, and MIOBI-MAKEEDGE, respectively, from Greedy to Adaptive, across various number of edges/nodes removal and edges addition and across different datasets. A non-negative entry in the table indicates that Adaptive performs better than Greedy; the higher of the number, the better the perform of Adaptive compare to Greedy. From these tables, the performance of our methods under Adaptive beats our methods under Greedy for any of the datasets and any of the manipulation we considered. Hence, for the later experiment, we will consider Adaptive for MIOBI-BREAKEDGE, MIOBI-BREAKNODE, and MIOBI-MAKEEDGE.

5.1 Competing Heuristic Strategies against Proposed MIOBI Framework

To show the effectiveness of our proposed methods, we compare our methods to other *competitive* strategies for each of the problem settings.

MIOBI-BREAKEDGE*: Edge Deletion* competing strategies. For edge deletion, we compare our method to the following 11 heuristic strategies.

1. **'rand'**: we randomly (with equal probability) select edges to remove [2];
2. **'rich-rich'**: for each edge (p, r), we compute the value of $d_p d_r$ and select the edges with the highest $d_p d_r$ to remove;
3. **'poor-poor'**: for each edge (p, r), we compute the value of $d_p d_r$ and select the edges with the lowest $d_p d_r$ to remove;

[2] The relative $\%$ change of robustness reported are averaged over 10 runs.

Table 4: MIoBI-BreakNode: % of relative robustness improvement from Greedy to Adaptive

Dataset	k=0.001%*n	k=0.005%*n	k=0.01%*n	k=0.015%*n	k=0.025%*n
O-A	0.00	0.00	0.00	1.07	3.18
O-B	9.88	5.63	4.30	8.60	5.38
O-C	0.00	2.56	4.78	4.18	6.45
O-D	0.00	8.87	9.94	14.52	12.15
O-E	0.00	6.72	10.55	12.80	15.21
O-F	0.00	6.11	8.14	6.40	9.12
O-G	8.99	5.23	5.93	8.76	11.28
O-H	11.03	4.29	8.27	8.53	10.66
O-I	3.30	10.19	10.58	13.02	15.51
P2P-A	0.31	3.90	43.55	4.82	7.70
P2P-B	0.27	0.71	5.11	7.24	10.16
P2P-C	53.13	54.27	25.88	22.40	18.44
P2P-D	16.08	2.61	7.68	10.69	15.14
P2P-E	11.53	15.90	23.98	26.96	37.50
W-V	0.21	0.89	3.43	5.56	9.63
E-E	0.29	28.14	47.33	54.11	47.79

Table 5: MIoBI-MakeEdge: % of relative robustness improvement from Greedy to Adaptive

Dataset	k=0.01%*n	k=0.05%*n	k=0.1%*n	k=0.15%*n	k=0.25%*n
O-A	0.00	0.00	3.29	2.19	2.95
O-B	0.00	2.46	4.13	5.78	8.59
O-C	0.00	3.69	6.51	7.83	20.34
O-D	0.00	2.53	1.69	1.17	2.95
O-E	0.00	3.77	4.44	5.93	12.95
O-F	0.45	3.74	9.22	15.81	34.39
O-G	0.38	2.19	6.67	15.90	31.67
O-H	0.63	3.19	9.15	19.46	34.92
O-I	0.73	2.85	9.80	22.08	36.25
P2P-A	1.00	7.82	7.78	4.61	9.39
P2P-B	0.48	7.69	2.84	7.09	11.00
P2P-C	1.49	7.63	14.57	16.50	13.77
P2P-D	3.21	6.65	7.39	9.82	6.87
P2P-E	3.88	23.76	22.08	18.17	12.97
W-V	0.12	0.42	0.55	0.84	1.67
E-E	0.40	1.99	7.73	12.96	18.15

4. **'rich-poor'**: for each edge (p, r), we compute the value of $|d_p - d_r|$ and select the edges with the highest $|d_p - d_r|$ to remove;
5. **'betw'**: for each edge (p, r), we compute the the edge-betweenness (the number of shortest paths between any pair of points that contain the edge (p, r)) and select the edges with the highest edge-betweenness;
6. **'embed'**: for each edge (p, r), we compute the edge-embeddedness (the number of common neighbors between p and r) and select the edges with the highest edge-embeddedness [18];

7. **'resist'**: for each edge (p, r), we compute the effective resistance of (p, r) (roughly speaking, it is equal to the probability of (p, r) appears in a random spanning tree and it measures the commute time between p and r), and select edges with the highest effective resistance [40];
8. **'netmelt'**: for each edge (p, r), we compute the (product of corresponding 1^{st} eigenvector values of p and r) value $\mathbf{u_{p1}u_{r1}}$ and select the edges with the highest $\mathbf{u_{p1}u_{r1}}$ [42];

   For the following heuristics, we use the concept of line graph. The line graph $L(G)$ of a graph G is a graph in which the nodes in $L(G)$ correspond to the edges in G and there an edge between two nodes in $L(G)$ if the corresponding edges in G incident to the same node (or to exactly one node) [42].

9. **'line-deg'**: for each (p, r) in G, we compute the degree of the corresponding node in $L(G)$ and select edges with the highest degree in $L(G)$;
10. **'line-eig'**: for each (p, r) in G, we compute the eigen-centrality (the first eigenvector value) of the corresponding node in $L(G)$ and select edges with the highest eigen-centrality in $L(G)$; and
11. **'line-page'**: for each (p, r), we compute the Pagerank score of the corresponding node in $L(G)$ and select edges with the highest Pagerank score in the line graph.

MIoBI-BreakNode*: Node Deletion* competing strategies. For node deletion, we compare our method to the following 5 heuristic strategies.

1. **'rand'**: we randomly (with equal probability) select nodes to remove[2];
2. **'max-deg'**: for each node i, we compute $n's$ degree and select the nodes with the highest degree;
3. **'eig'**: for each node i, we compute its eigen-centrality (a value that is based on the 1^{st} eigenvector value) and select the nodes with the highest eigen-centrality;
4. **'page'**: for each node i, we compute $i's$ Pagerank score and select nodes with the highest Pagerank score; and
5. **'cluster'**: for each node i, we compute $i's$ local clustering coefficient, the fraction of the number of edges exists among $i's$ neighbors and the number of *possible* edges that can exist among $i's$ neighbors, and select the nodes with the highest local clustering coefficient.

MIoBI-MakeEdge*: Edge Addition* competing strategies. For node addition, we compare our method to the following 5 heuristic strategies.

1. **'rand'**: we randomly (with equal probability) select k nonexistent edges to add[2];
2. **'rich-rich'**: for each nonexistent edge (p, r), we compute the value of $d_p d_r$ and select the edges with the highest degree;
3. **'poor-poor'**: for each nonexistent edge (p, r), we compute the value of $d_p d_r$ and select the edges with the lowest degree (same as 'preferential addition' in [3]);
4. **'rich-poor'**: for each nonexistent edge (p, r), we compute the value of $|d_p - d_r|$ and select the edges with highest $|d_p - d_r|$;
5. **'netgel'**: for each nonexistent edge (p, r), we compute the (product of corresponding 1^{st} eigenvector values of p and r) value $\mathbf{u_{p1}u_{r1}}$ and select the edges with the highest $\mathbf{u_{p1}u_{r1}}$ [42];

5.2 Effectiveness of Proposed MIoBI Framework

Figure 1, 2, and 3 show the performance results of % relative robustness change vs. k for all methods all datasets of MIoBI-BreakEdge, MIoBI-BreakNode, and MIoBI-MakeEdge, respectively. Using RC@50, our methods outperform all other competitive strategies . Although our 'Naive' methods did not perform as well as other strategies due to the accumulation of updating errors, it still outperform more than half of the competitive strategies we introduced.

For edge removal, our method MIoBI-BreakEdge RC@50 achieves higher % of relative robustness change than other competitive strategies across all the datasets we considered. Among the strategies, 'rich-rich' obtains the second highest % of relative robustness change.

For node removal, our method MIoBI-BreakNode RC@50 achieves one of the highest % of relative robustness change among the competing strategies across all the datasets we considered. Among the strategies, 'max-deg' % of relative robustness change comes very close to our method.

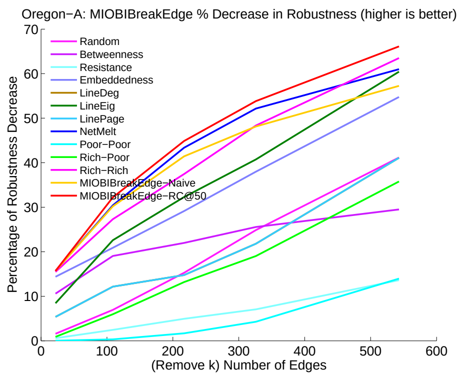
For edge addition, our method MIoBI-MakeEdge RC@50 achieves the highest % of relative robustness change than other competitive strategies across all the datasets we considered. None of the other strategies comes close to ours and our methods outperforms all of them significantly.

5.3 Scalability of Proposed MIoBI Framework

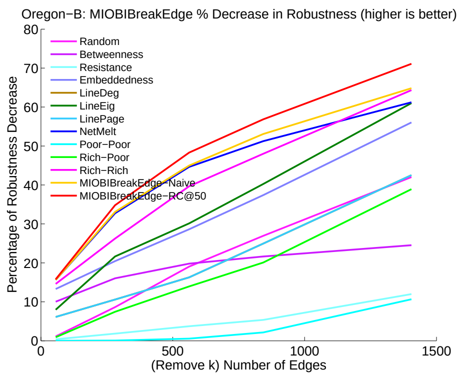
We use the *Oregon A-I* datasets, sorted by the number of edges m, to evaluate the scalability of the proposed algorithms for growing graph sizes. The run time results are presented in Figure 4 for various k.[3]

We observe that the proposed methods empirically scale near-linearly wrt m, which demonstrates that they are suitable for large graphs. In particular, for MIoBI-BreakEdge and MIoBI-MakeEdge, the computation times of Naive and RC@50 are very close to each other as is evident from Figure 4 (a) and (c) across different k values, respectively for edge removal and edge addition. Both approaches are quite efficient for large graphs. On the other hand, the result is different for MIoBI-BreakNode. In particular, Figure 4 (b) shows that the gap of computation times between Naive and RC@50 are larger than before for different values of k. We attribute this behavior to the amount of perturbation caused by node removals; as a result of each node that we remove, we also remove the edges that are attached to it, the count of which often exceeds 50. Therefore, we essentially perform updates at every step with RC@50, and hence its larger computation time. Note that the scalability, on the other hand, remains near-linear at all cases.

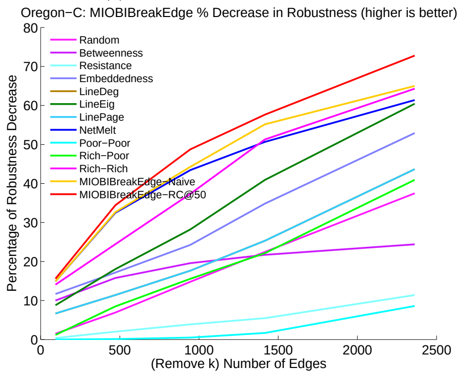
[3] All reported times are on a 64-bit machine, Intel Core i5-3570K CPU @3.40GHz and 8GB memory, running Ubuntu 12.10.



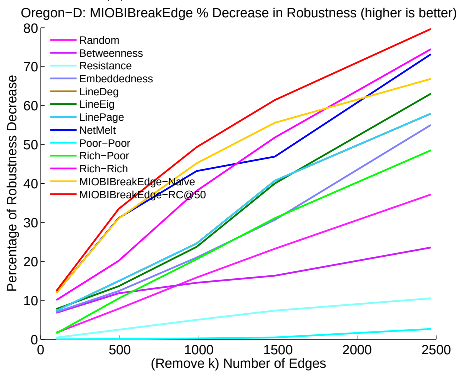
(a) O-A: MIOBI-BREAKEDGE



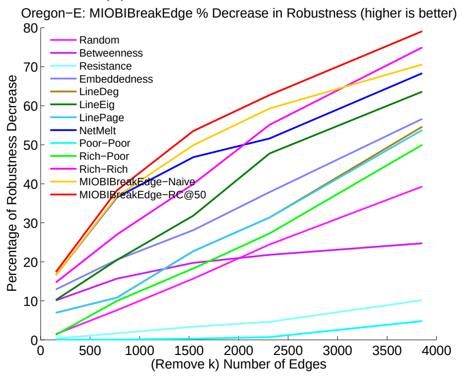
(b) O-B: MIOBI-BREAKEDGE



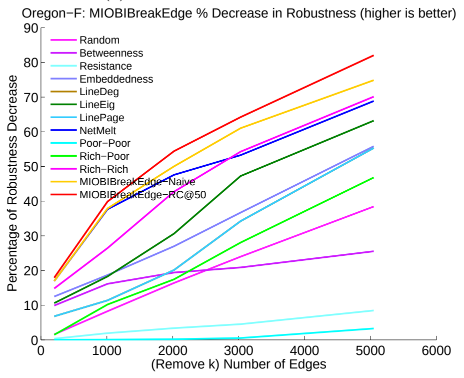
(b) O-C: MIOBI-BREAKEDGE



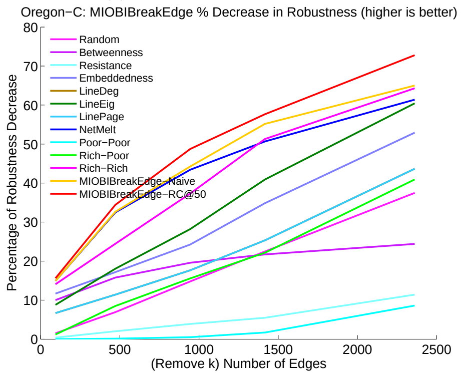
(c) O-D: MIOBI-BREAKEDGE



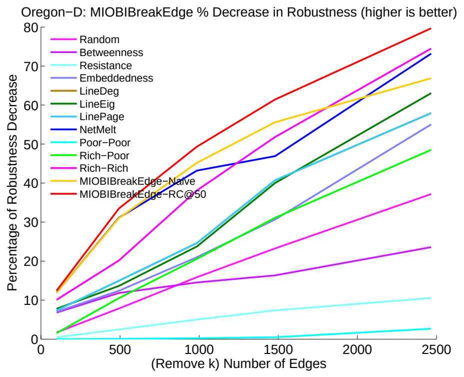
(e) O-E: MIOBI-BREAKEDGE



(f) O-F: MIOBI-BREAKEDGE

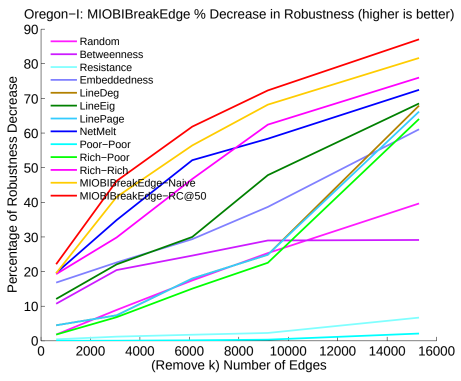


(g) O-G: MIOBI-BREAKEDGE

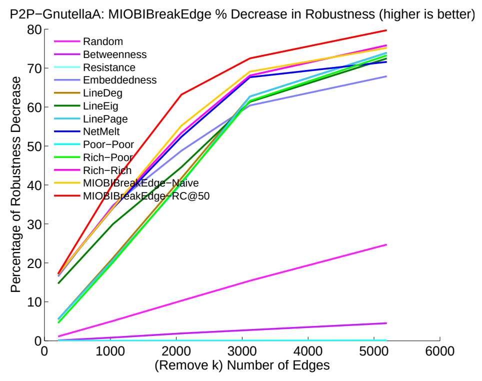


(h) O-H: MIOBI-BREAKEDGE

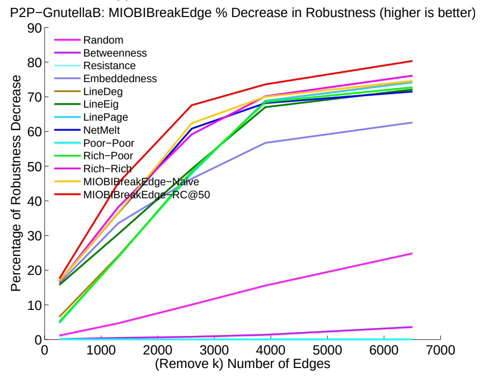
Fig. 1: % robustness change (higher is better) vs. k for MIOBI-BREAKEDGE and various heuristics on all datasets. Notice that our methods outperform all the heuristics. (figures best in color)



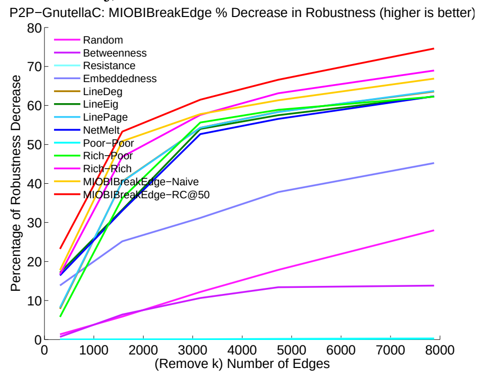
(i) O-I: MIOBI-BREAKEDGE



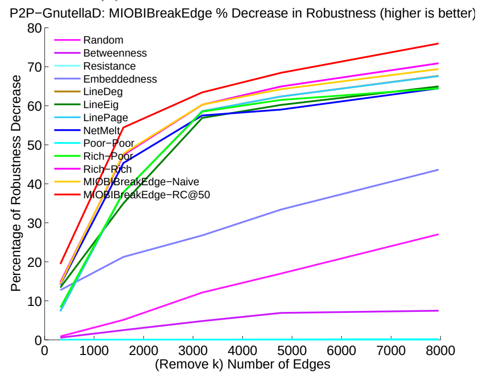
(j) P2P-A: MIOBI-BREAKEDGE



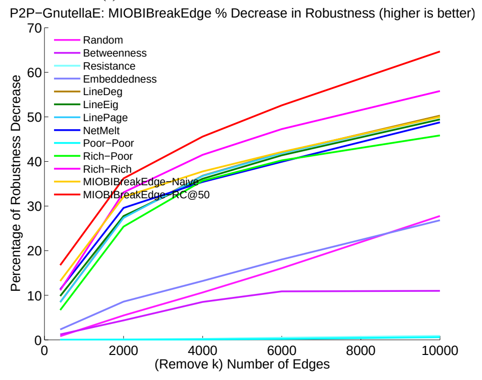
(k) P2P-B: MIOBI-BREAKEDGE



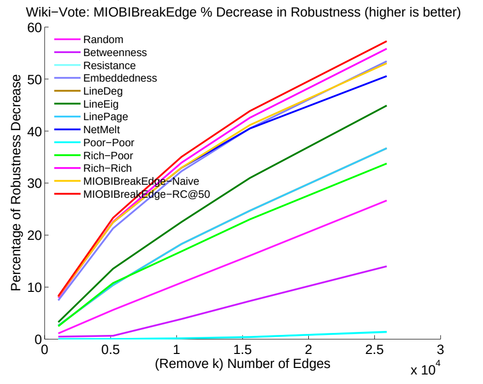
(l) P2P-C: MIOBI-BREAKEDGE



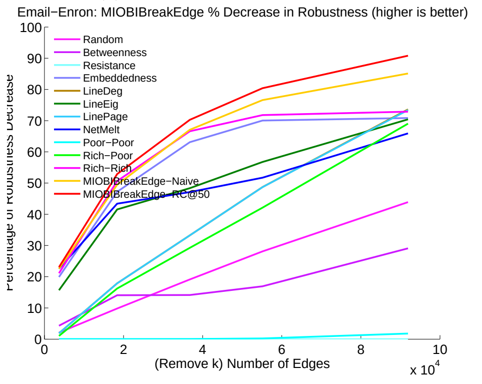
(m) P2P-D: MIOBI-BREAKEDGE



(n) P2P-E: MIOBI-BREAKEDGE

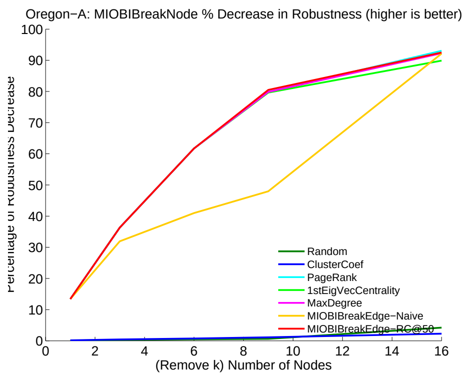


(o) W-V: MIOBI-BREAKEDGE

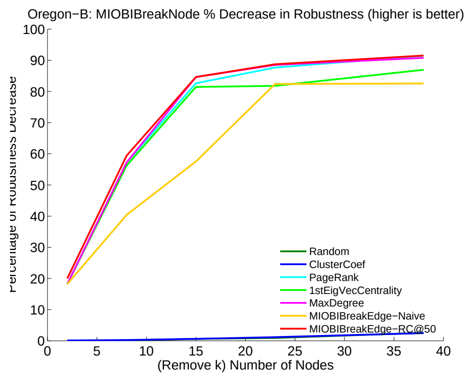


(p) E-E: MIOBI-BREAKEDGE

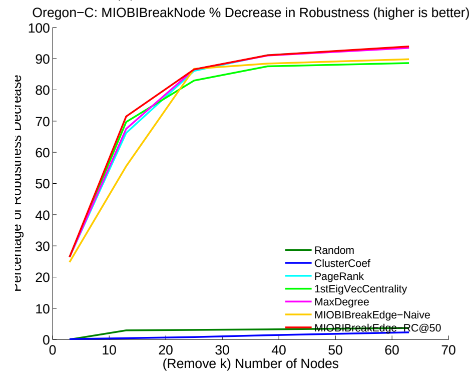
Fig. 1: % robustness change (higher is better) vs. k for MIOBI-BREAKEDGE and various heuristics on all datasets. Notice that our methods outperform all the heuristics. (figures best in color)



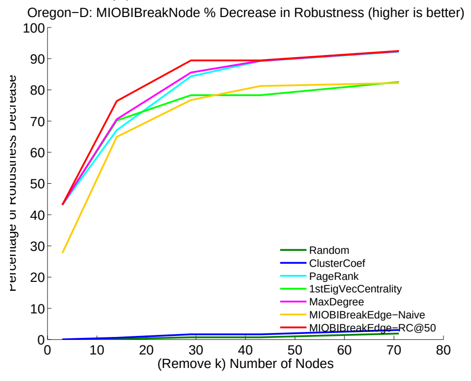
(a) O-A: MIOBI-BREAKNODE



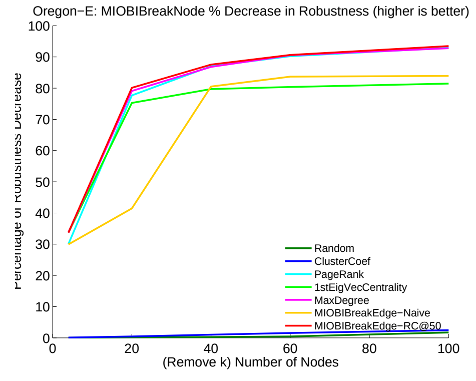
(b) O-B: MIOBI-BREAKNODE



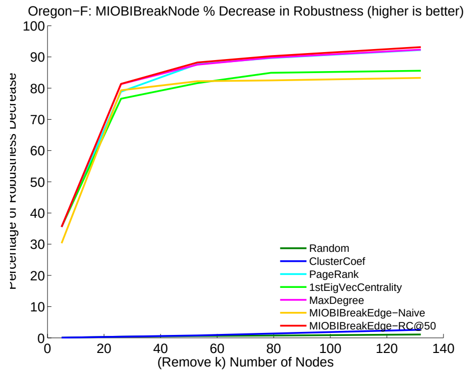
(b) O-C: MIOBI-BREAKNODE



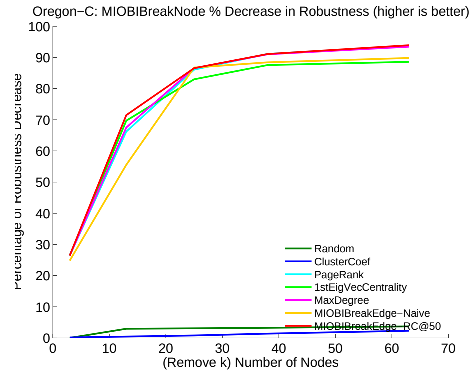
(c) O-D: MIOBI-BREAKNODE



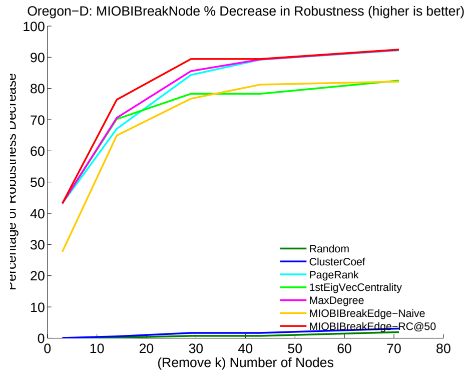
(e) O-E: MIOBI-BREAKNODE



(f) O-F: MIOBI-BREAKNODE

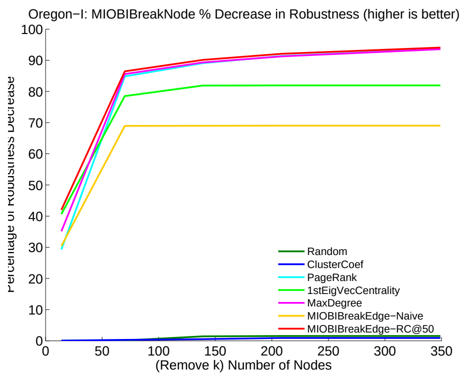


(g) O-G: MIOBI-BREAKNODE

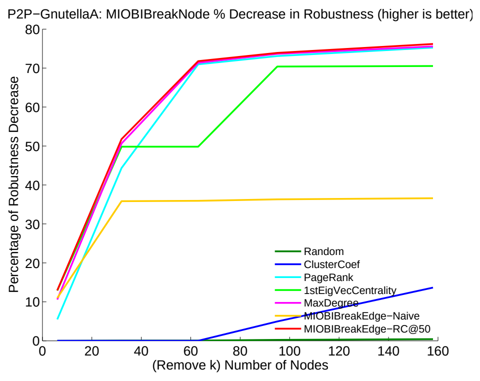


(h) O-H: MIOBI-BREAKNODE

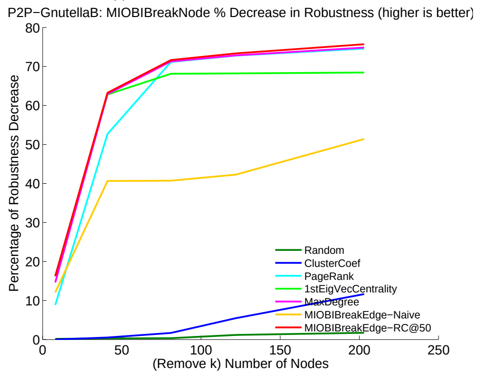
Fig. 2: % robustness change (higher is better) vs. k for MIOBI-BREAKNODE and various heuristics on all datasets. Notice that our methods outperform all the heuristics. (figures best in color)



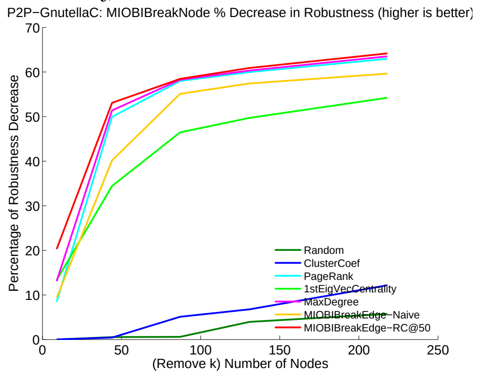
(i) O-I: MIOBI-BREAKNODE



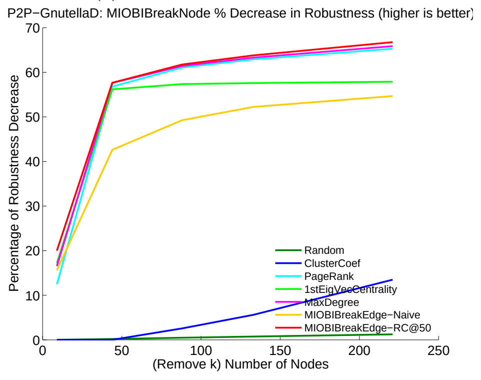
(j) P2P-A: MIOBI-BREAKNODE



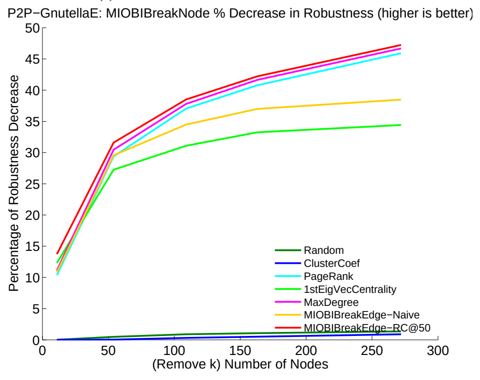
(k) P2P-B: MIOBI-BREAKNODE



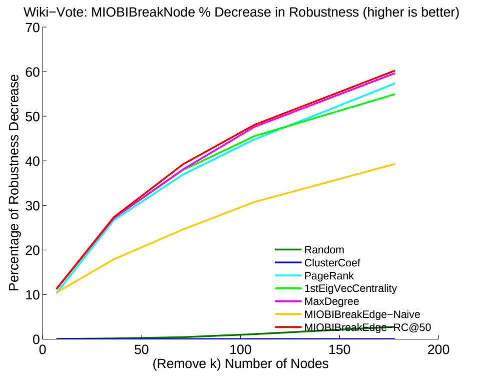
(l) P2P-C: MIOBI-BREAKNODE



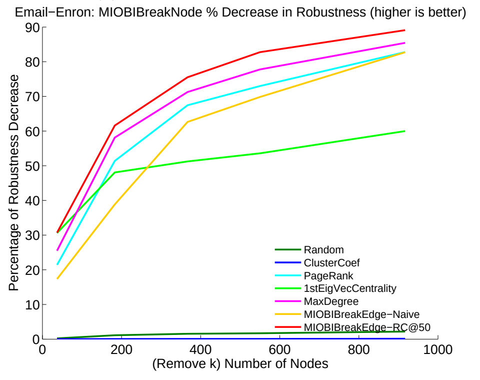
(m) P2P-D: MIOBI-BREAKNODE



(n) P2P-E: MIOBI-BREAKNODE

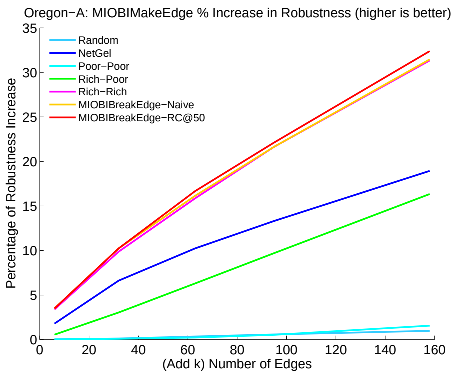


(o) W-V: MIOBI-BREAKNODE

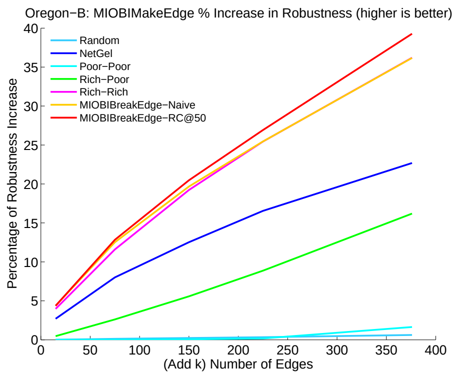


(p) E-E: MIOBI-BREAKNODE

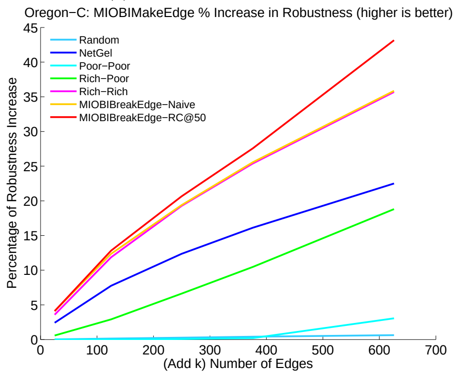
Fig. 2: % robustness change (higher is better) vs. k for MIOBI-BREAKNODE and various heuristics on all datasets. Notice that our methods outperform all the heuristics. (figures best in color)



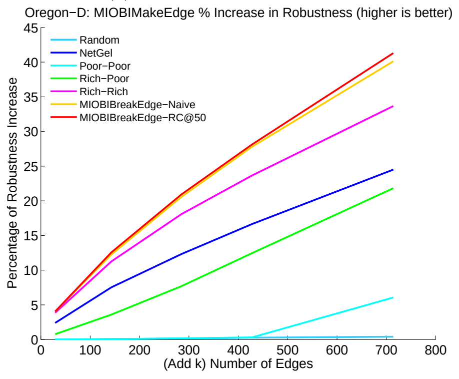
(a) O-A: MIOBI-MAKEEDGE



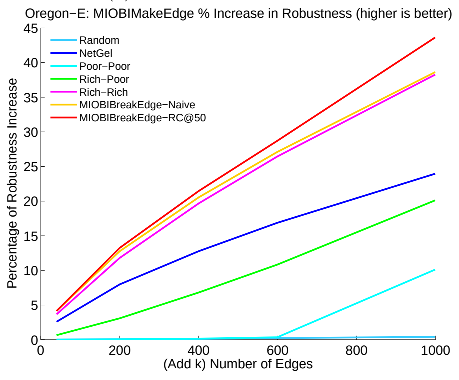
(b) O-B: MIOBI-MAKEEDGE



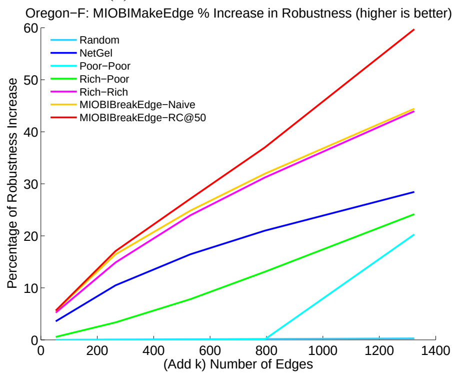
(b) O-C: MIOBI-MAKEEDGE



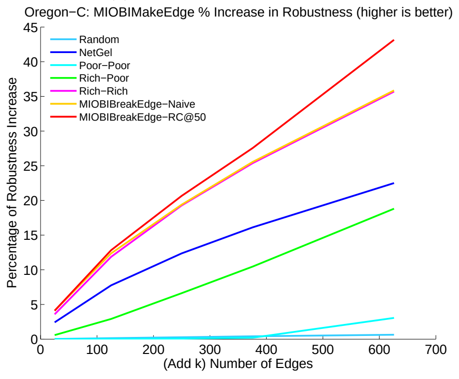
(c) O-D: MIOBI-MAKEEDGE



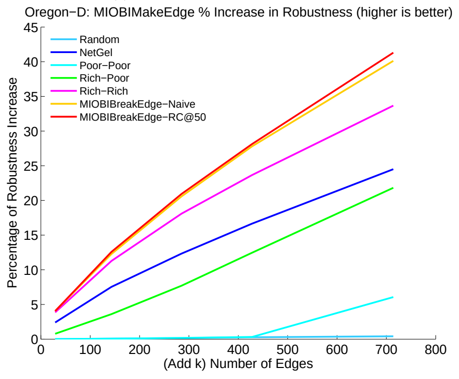
(e) O-E: MIOBI-MAKEEDGE



(f) O-F: MIOBI-MAKEEDGE

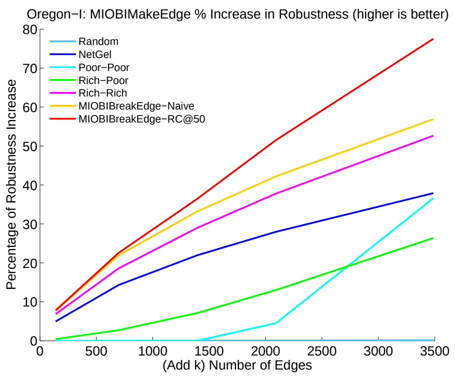


(g) O-G: MIOBI-MAKEEDGE

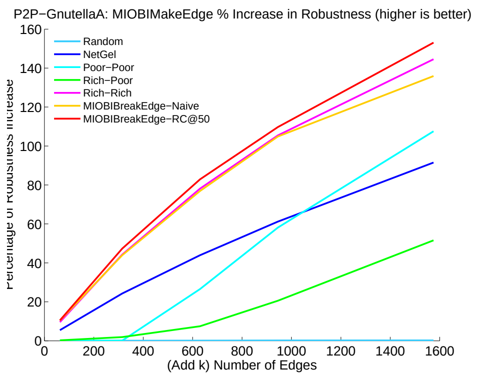


(h) O-H: MIOBI-MAKEEDGE

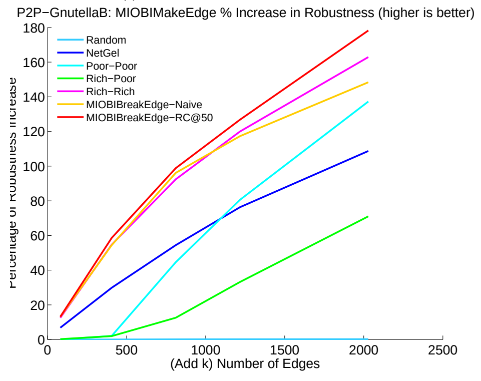
Fig. 3: % robustness change (higher is better) vs. k for MIOBI-MAKEEDGE and various heuristics on all datasets. Notice that our methods outperform all the heuristics. (figures best in color)



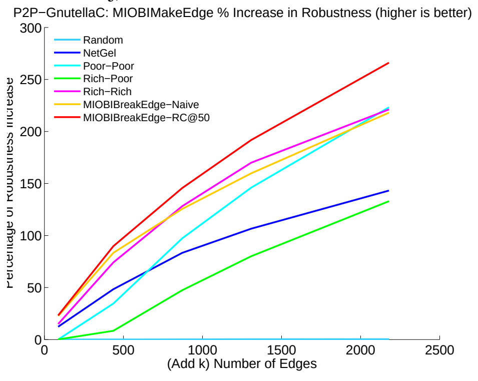
(i) O-I: MIOBI-MAKEEDGE



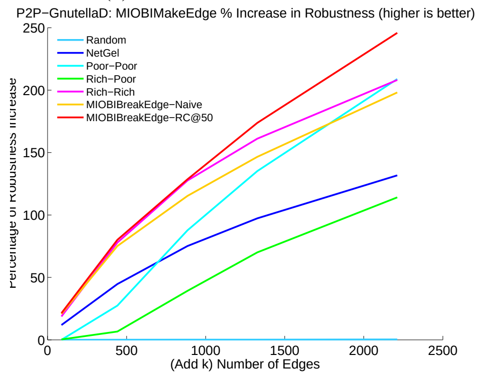
(j) P2P-A: MIOBI-MAKEEDGE



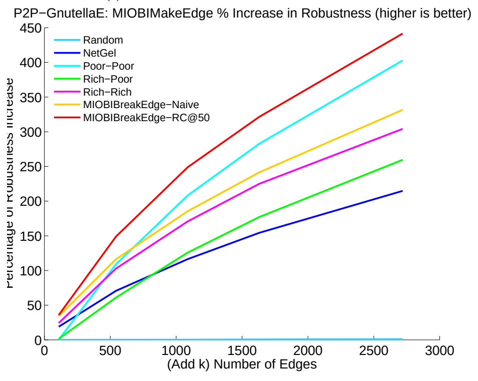
(k) P2P-B: MIOBI-MAKEEDGE



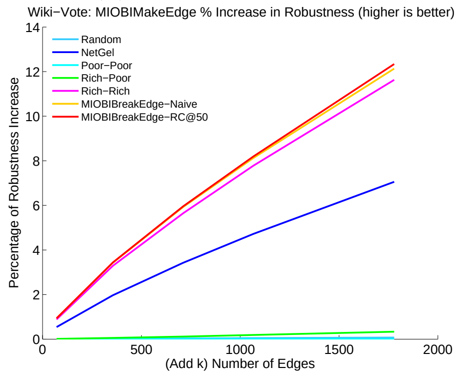
(l) P2P-C: MIOBI-MAKEEDGE



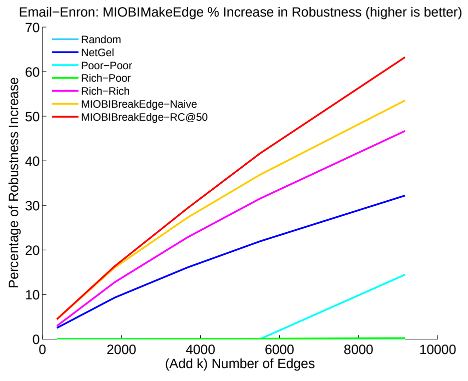
(m) P2P-D: MIOBI-MAKEEDGE



(n) P2P-E: MIOBI-MAKEEDGE

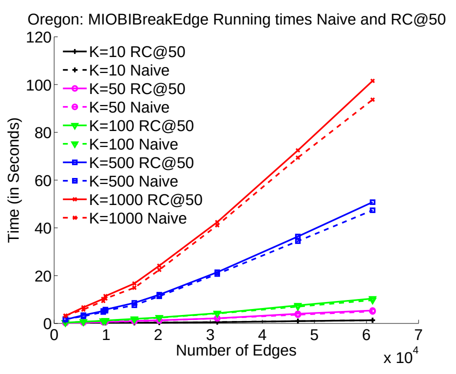


(o) W-V: MIOBI-MAKEEDGE

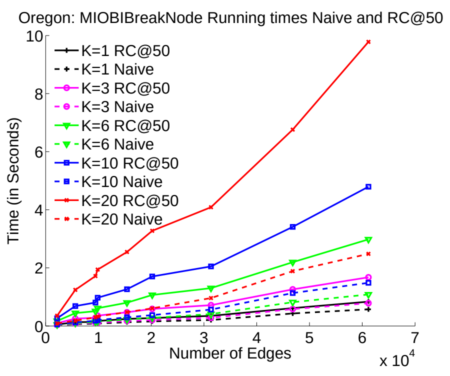


(p) E-E: MIOBI-MAKEEDGE

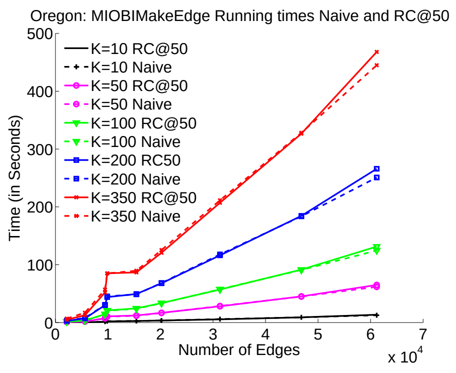
Fig. 3: % robustness change (higher is better) vs. k for MIOBI-MAKEEDGE and various heuristics on all datasets. Notice that our methods outperform all the heuristics. (figures best in color)



(a) MIOBI-BREAKEDGE



(b) MIOBI-BREAKNODE



(c) MIOBI-MAKEEDGE

Fig. 4: Scalability of proposed methods—all three algorithms scale near-linearly wrt graph size. (figures best viewed in color)

6 Conclusion

In this work we studied graph robustness where we focused on two main problems; (i) how to define and measure robustness of a graph topology, and (ii) how to maximally and efficiently alter the robustness for large-scale graphs.

We first considered various definitions and measures of robustness, and analyzed their capabilities and shortcomings of capturing desired resilience properties of graphs. We identified natural connectivity as a reliable measure, as it accounts for the existence of alternative paths in a graph, effectively quantifies their count and lengths, changes strictly monotonically by edge additions, etc. Later, we formulated two new robustness manipulation problems, one of which is to maximally decrease (or "break") the robustness with edge or node deletions, and another is to maximally improve (or "make") the robustness with edge additions. We studied and showed the hardness associated with these problems, and proposed effective, scalable, and adaptive algorithms to solve them, which are founded on a principled framework based on theoretical foundations. Finally, our experiments showed the superiority of our methods compared to a long list of heuristic, ad-hoc strategies.

We remark that applications from diverse domains of networked systems are likely to benefit from our analysis and algorithms proposed in this work. For example, network administrators are often involved with measuring and optimizing the performance on infrastructure networks such as the Internet, the power-grid, road/airline networks, and other transmission networks (e.g., oil, gas, etc.). Similarly, socially connected networks, such as Twitter, Facebook, the blogosphere, as well as communities at smaller scales (e.g., school children), are often monitored by policy makers and other entities who are concerned with optimizing or controlling the efficiency of spread (e.g., information, disease, rumor, etc.) on these networks.

References

1. Albert, R., Jeong, H., Barabasi, A.L.: Error and attack tolerance of complex networks. Nature **406**(6794), 378–382 (2000)
2. Barefoot, C.A., Entringer, R., Swart, H.: Integrity of trees and powers of cycles. Eighteenth Southeastern International Conference on Combinatorics, Graph Theory, and Computing **58**, 103–114 (1987)
3. Beygelzimer, A., Grinstein, G., Linsker, R., Rish, I.: Improving network robustness by edge modification. Physica A: Statistical Mechanics and its Applications **357**(3-4), 593–612 (2005)
4. Callaway, D.S., Newman, M.E.J., Strogatz, S.H., Watts, D.J.: Network Robustness and Fragility: Percolation on Random Graphs. Physical Review Letters **85**(25), 5468–5471 (2000)
5. Choi, H.A., Esfahanian, A.H.: Restricted vertex-connectivity with application to fault-tolerant computing. In: Proceedings of The International Conference on Operations Research (1990)
6. Chvátal, V.: Tough graphs and hamiltonian circuits. Discrete Mathematics **306**(10-11), 910–917 (2006)
7. Cohen, R., Erez, K., Avraham, D.B., Havlin, S.: Breakdown of the Internet under Intentional Attack. Physical Review Letters **86**(16), 3682–3685 (2001). DOI 10.1103/physrevlett.86.3682
8. Crucitti, P., Latora, V., Marchiori, M., Rapisarda, A.: Error and attack tolerance of complex networks. Physica A: Statistical Mechanics and its Applications (1-3), 388–394 (2004)
9. Dinh, T.N., Xuan, Y., Thai, M.T., Pardalos, P.M., Znati, T.: On new approaches of assessing network vulnerability: Hardness and approximation. IEEE/ACM Trans. Netw. **20**(2), 609–619 (2012)
10. Estrada, E.: Characterization of 3D molecular structure. Chemical Physics Letters **319**(5-6), 713–718 (2000)
11. Estrada, E.: Network robustness to targeted attacks. the interplay of expansibility and degree distribution. The European Physical Journal B - Condensed Matter and Complex Systems **52**(4), 563–574 (2006)
12. Estrada, E.: Network robustness to targeted attacks. the interplay of expansibility and degree distribution. The European Physical Journal B - Condensed Matter and Complex Systems **52**(4), 563–574 (2006). DOI 10.1140/epjb/e2006-00330-7
13. Estrada, E., Hatano, N., Benzi, M.: The physics of communicability in complex networks. CoRR **abs/1109.2950** (2011)
14. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. SIGCOMM Comput. Commun. Rev. **29**(4), 251–262 (1999). DOI 10.1145/316194.316229. URL `http://doi.acm.org/10.1145/316194.316229`
15. Fiedler, M.: Algebraic Connectivity of Graphs. Czechoslovak Mathematical Journal **23**, 298–305 (1973)
16. Frank, H., Frisch, I.: Analysis and Design of Survivable Networks. Communication Technology, IEEE Transactions on **18**(5), 501–519 (1970). DOI 10.1109/tcom.1970.1090419
17. Gantmacher, F.: The Theory of Matrices. No. Bd. 2 in Chelsea Publishing. American Mathematical Society (1960)
18. Granovetter, M.S.: Economic action and social structure: the problem of embeddedness. American Journal of Sociology **91**, 481–510 (1985)

19. Holme, P., Kim, B.J., Yoon, C.N., Han, S.K.: Attack vulnerability of complex networks. Physical Review E **65**(5), 056,109 (2002)
20. Jung, H.A.: On a class of posets and the corresponding comparability graphs. J. Comb. Theory, Ser. B **24**(2), 125–133 (1978)
21. Kleinberg, J.M., Sandler, M., Slivkins, A.: Network failure detection and graph connectivity. In: J.I. Munro (ed.) SODA, pp. 76–85. SIAM (2004)
22. Krishnamoorthy, M., Krishnamurthy, B.: Fault diameter of interconnection networks. Computers & Mathematics with Applications **13**(5–6), 577 – 582 (1987)
23. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Nat. Bur. Stand. **45**, 255–82 (1950)
24. Louzada, V.H.P., Daolio, F., Herrmann, H.J., Tomassini, M.: Smart rewiring for network robustness. CoRR **abs/1303.5269** (2013)
25. M. Cozzens, D.M., Stueckle, S.: The tenacity of a graph. Proc. 7th International Conference on the Theory and Applications of Graphs **24**, 1111–1122 (1995)
26. Malliaros, F.D., Megalooikonomou, V., Faloutsos, C.: Fast robustness estimation in large social graphs: Communities and anomaly detection. In: SDM, pp. 942–953 (2012)
27. Meyer, C.D. (ed.): Matrix analysis and applied linear algebra. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000)
28. Mieghem, P.V., Wang, H., Ge, X., Tang, S., Kuipers, F.A.: Influence of assortativity and degree-preserving rewiring on the spectra of networks. The European Physical Journal B - Condensed Matter and Complex Systems **76**(4), 643–652 (2010)
29. Mohar, B.: Isoperimetric numbers of graphs. J. Comb. Theory, Ser. B **47**(3), 274–291 (1989)
30. Newman, M.E.J.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. USA **103**, 8577 (2006)
31. Nguyen, N.P., Alim, M.A., Shen, Y., Thai, M.T.: Assessing network vulnerability in a community structure point of view. In: ASONAM, pp. 231–235. ACM (2013). URL `http://dblp.uni-trier.de/db/conf/asunam/asonam2013.html#NguyenAST13`
32. Nikiforov, V.: Some inequalities for the largest eigenvalue of a graph. Combinatorics, Probability & Computing **11**(2), 179–189 (2002)
33. Pan, V.Y., Chen, Z.Q.: The complexity of the matrix eigenproblem. In: J.S. Vitter, L.L. Larmore, F.T. Leighton (eds.) STOC, pp. 507–516. ACM (1999)
34. Paul, G., Tanizawa, T., Havlin, S., Stanley, H.: Optimization of robustness of complex networks. The European Physical Journal B - Condensed Matter and Complex Systems **38**(2), 187–191 (2004)
35. Scellato, S., Leontiadis, I., Mascolo, C., Basu, P., Zafer, M.: Evaluating temporal robustness of mobile networks. IEEE Trans. Mob. Comput. **12**(1), 105–117 (2013)
36. Schneider, C.M., Moreira, A.A., Jr., J.S.A., Havlin, S., Herrmann, H.J.: Mitigation of malicious attacks on networks. CoRR **abs/1103.1741** (2011)
37. Shang, Y.L.: Local natural connectivity in complex networks. Chinese Physics Letters **28**(6) (2011)
38. Shargel, B., Sayama, H., Epstein, I.R., Bar-Yam, Y.: Optimization of robustness and connectivity in complex networks. Phys Rev Lett **90**(6), 068,701 (2003)
39. Shen, Y., Nguyen, N.P., Xuan, Y., Thai, M.T.: On the discovery of critical links and nodes for assessing network vulnerability. IEEE/ACM Trans. Netw. **21**(3), 963–973 (2013). URL `http://dblp.uni-trier.de/db/journals/ton/ton21.html#ShenNXT13`
40. Spielman, D.A., Srivastava, N.: Graph sparsification by effective resistances. SIAM J. Comput. **40**(6), 1913–1926 (2011)
41. Stewart, G.W., Sun, J.G.: Matrix Perturbation Theory. Academic Press (1990)
42. Tong, H., Prakash, B.A., Eliassi-Rad, T., Faloutsos, M., Faloutsos, C.: Gelling, and melting, large graphs by edge manipulation. In: CIKM, pp. 245–254. ACM (2012)
43. Tsourakakis, C.E.: Fast counting of triangles in large real networks without counting: Algorithms and laws. In: ICDM, pp. 608–617. IEEE Computer Society (2008)
44. Van Mieghem, P., Stevanovic, D., Kuipers, F., Li, C., van de Bovenkamp, R., Liu, D., Wang, H.: Decreasing the spectral radius of a graph by link removals (2011). URL `http://dx.doi.org/10.1103/PhysRevE.84.016101`
45. Wu, J., Mauricio, B., Tan, Y.J., Deng, H.Z.: Natural connectivity of complex networks. Chinese Physics Letters **27**(7), 78902 (2010). DOI 10.1088/0256-307X/27/7/078902