Evaluating Graph Vulnerability and Robustness using TIGER

Scott Freitas
safreita@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia

Duen Horng Chau
polo@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia

ABSTRACT

The study of network robustness is a critical tool in the characterization and understanding of complex interconnected systems such as transportation, infrastructure, communication, and computer networks. Through analyzing and understanding the robustness of these networks we can: (1) quantify network vulnerability and robustness, (2) augment a network's structure to resist attacks and recover from failure, and (3) control the dissemination of entities on the network (e.g., viruses, propaganda). While significant research has been conducted on all of these tasks, no comprehensive open-source toolbox currently exists to assist researchers and practitioners in this important topic. This lack of available tools hinders reproducibility and examination of existing work, development of new research, and dissemination of new ideas. We contribute Tiger, an open-sourced Python toolbox to address these challenges. Tiger contains 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization based defense techniques; and 4 simulation tools. By democratizing the tools required to study network robustness, our goal is to assist researchers and practitioners in analyzing their own networks; and facilitate the development of new research in the field. Tiger is open-sourced at: https://github.com/safreita1/TIGER.

CCS CONCEPTS

• **Information systems** → Social networks; **Computing platforms**.

KEYWORDS

Graphs, robustness, vulnerability, networks, attacks, defense

1 INTRODUCTION

Motivation. First mentioned as early as the 1970's [7], network robustness has a rich and diverse history spanning numerous fields of engineering and science [3, 14, 23, 26, 30]. This diversity of research has generated a variety of unique perspectives, providing fresh insight into challenging problems while equipping researchers with fundamental knowledge for their investigations. While the fields of study are diverse, they are linked by a common definition of **robustness**, which is defined as a *measure of a network's ability to continue functioning when part of the network is naturally damaged or targeted for attack* [3, 5, 9].

The study of network robustness is critical to the understanding of complex interconnected systems. For example, consider an example of a power grid network that is susceptible to both natural *failures* and targeted *attacks*. A natural failure occurs when a *single* power substation fails due to erosion of parts or natural disasters. However, when one substation fails, additional load is routed to alternative substations, potentially causing a series of *cascading failures*. Not all failures originate from natural causes, some come from *targeted* attacks, such as enemy states hacking into the grid to sabotage key equipment to maximally damage the operations of the electrical grid. A natural counterpart to network robustness is **vulnerability**, defined as a *measure of a network's susceptibility to the dissemination of entities across the network* [30], such as how quickly a virus spreads across a computer network.

Challenges for robustness and vulnerability research. Unfortunately, the nature of cross-disciplinary research also comes with significant challenges. Oftentimes important discoveries made in one field are not quickly disseminated, leading to missed innovation opportunities. We believe a unified and easy-to-use software framework is key to standardizing the study of network robustness, helping accelerate reproducible research and dissemination of ideas.

Tiger design and implementation. We present Tiger, an open-sourced Python ***T**oolbox for evaluat**I**ng **G**raph vuln**E**rability and **R**obustness*. Through Tiger, our goal is to catalyze network robustness research, promote reproducibility and amplify the reach of novel ideas. In designing Tiger, we consider multiple complex implementation decisions, including: (1) the criterion for inclusion in the toolbox; (2) identifying and synthesizing a set of core robustivity features needed by the community; and (3) the design and implementation of the framework itself. We address the *inclusion criterion* by conducting a careful analysis of influential and representative papers (e.g., [3, 5, 19, 28, 30]) across top journals and conferences from the relevant domains (e.g., ICDM, SDM, Physica A, DMKD, Physical Review E), many of which we will discuss in detail in this paper. We also include papers posted on arXiv, as many cutting-edge papers are first released here.

Based on our analysis, we identify and include papers that tackle one or more of the following fundamental tasks [3, 9, 19]: (1) measuring network robustness and vulnerability; (2) understanding network failure and attack mechanisms; (3) developing defensive techniques; and (4) creating simulation tools to model processes. From these papers, we select and implement a total of 44 attacks, defenses and robustness measures, along with 4 simulation tools in which they can be used. Due to a vibrant and growing community of users, we develop Tiger in Python 3, leveraging key libraries,

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

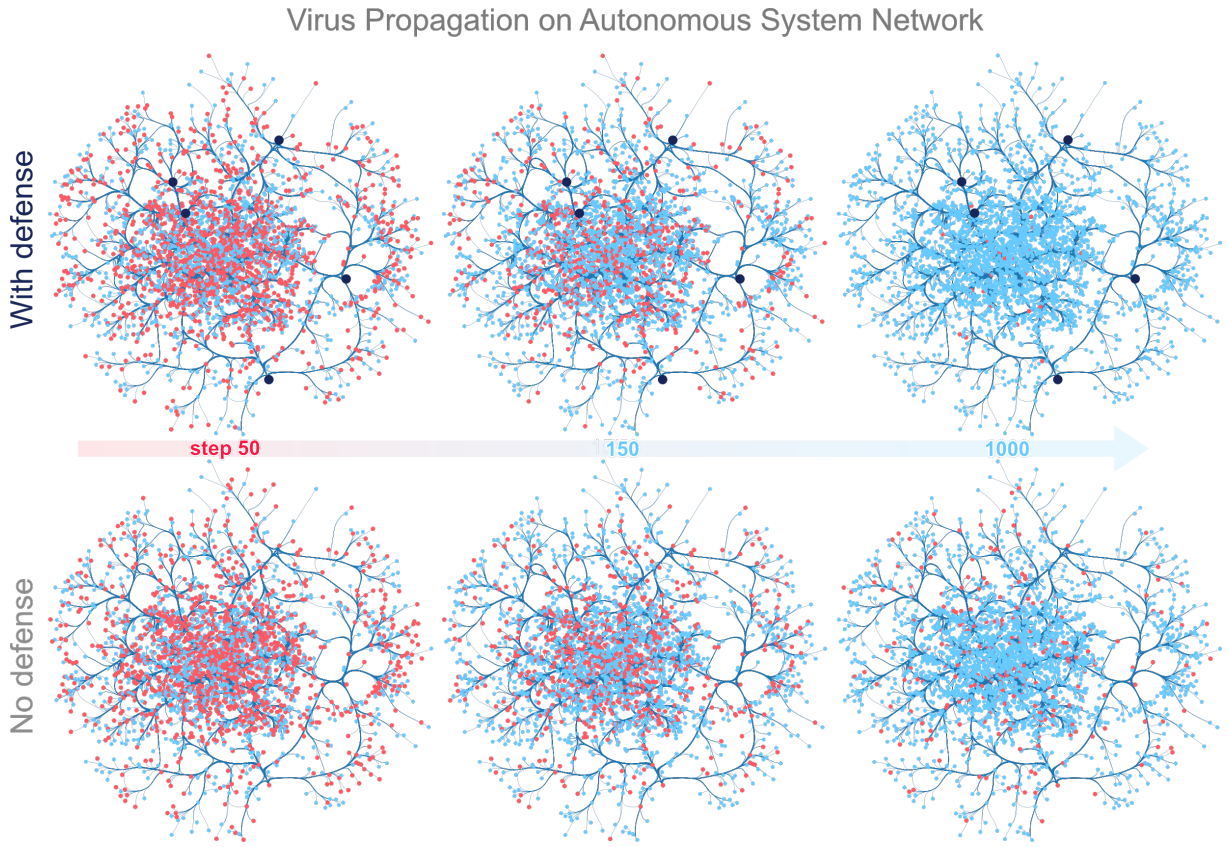

Figure 1: Tiger provides a number of important tools for graph vulnerability and robustness research, including graph robustness measures, attack strategies, defense techniques and simulation models. Here, a Tiger user is visualizing a computer virus simulation that follows the SIS infection model (effective strength $s = 3.21$) on the *Oregon-1 Autonomous System* network [27]. Top: defending only 5 nodes with *Netshield* [30], the number of infected entities is reduced to nearly zero. Bottom: without any defense, the virus remains endemic.

such as NetworkX, SciPy, Numpy and Matplotlib. While excellent alternative network analysis tools exist [1, 2, 13, 16, 21, 24, 25, 29], many of them are domain specific (e.g., EoN [1], WNTR [24]) or do not provide direct support for network robustness analysis (e.g., NetworkX [16], Gephi [2]). In contrast, Tiger complements existing tools while providing key missing network robustness components.

1.1 Contributions

1. TIGER. We present Tiger, the first open-sourced Python toolbox for evaluating network vulnerability and robustness of graphs. Tiger contains 22 graph robustness measures with both original and fast approximate versions when possible; 17 failure and attack mechanisms; 15 heuristic and optimization based defense techniques; and 4 simulation tools. To the best of our knowledge, this makes Tiger the most comprehensive open-source framework for network robustness analysis to date.

2. Open-Source & Permissive Licensing. Our goal is to democratize the tools needed to study network robustness; assisting researchers and practitioners in the analysis of their own networks. As such, we open-source the *code* on Github with an *MIT license*, available at: https://github.com/safreita1/TIGER.

3. Extensive Documentation & Tutorials. We extensively document the functionality of Tiger, providing a detailed description and working example for many robustness measures, attacks, and defense mechanisms. In addition, we provide detailed tutorials on the analysis of network vulnerability and robustness on multiple large-scale, real-world networks, including *every* figure and plot shown in this paper. Users with Python familiarity will be able to readily pick up Tiger for analysis with their own data.

4. Community Impact. Tiger helps enable reproducible research and the timely dissemination of new and current ideas in the area of network robustness and vulnerability analysis. Since this is a *large* and *highly active* field across many disciplines of science and engineering, we anticipate that Tiger will have significant impact. As the field grows, we will update Tiger with additional features such as multi-graphs, dynamic networks and rich attribute information.

Robustness Measure	Category	Application to Network Robustness				
Vertex connectivity	graph	higher value	⇒	harder to disconnect graph	⇒	higher robustness
Edge connectivity	graph	higher value	⇒	harder to disconnect graph	⇒	higher robustness
Diameter	graph	lower value	⇒	stronger connectivity	⇒	higher robustness
Average distance	graph	lower value	⇒	stronger connectivity	⇒	higher robustness
Average inverse distance	graph	higher value	⇒	stronger connectivity	⇒	higher robustness
Average vertex betweenness	graph	lower value	⇒	more evenly distributed load	⇒	higher robustness
Average edge betweenness	graph	lower value	⇒	more evenly distributed load	⇒	higher robustness
Global clustering coefficient	graph	higher value	⇒	more triangles	⇒	higher robustness
Largest connected component	graph	lower value	⇒	more disconnected graph	⇒	lower robustness
Spectral radius	adjacency	larger value	⇒	stronger connectivity	⇒	higher robustness
Spectral gap	adjacency	higher value	⇒	fewer bottlenecks	⇒	higher robustness
Natural connectivity	adjacency	higher value	⇒	more alternative pathways	⇒	higher robustness
Spectral scaling	adjacency	lower value	⇒	fewer bottlenecks	⇒	higher robustness
Generalized robustness index	adjacency	lower value	⇒	fewer bottlenecks	⇒	higher robustness
Algebraic connectivity	laplacian	higher value	⇒	harder to disconnect	⇒	higher robustness
Number of spanning trees	laplacian	higher value	⇒	more alternative pathways	⇒	higher robustness
Effective resistance	laplacian	lower value	⇒	more alternative pathways	⇒	higher robustness

Table 1: Comparison of Tiger robustness measures. Measures are grouped based on whether they use the *graph, adjacency* or *Laplacian* matrix. For each measure, we briefly describe it's application to measuring network robustness. We omit approximate measures since their application to robustness is the same as the non-approximate counterparts.

Terminology and Notation. As the study of graphs has been carried out in a variety of fields (e.g., mathematics, physics, computer science), the terminology often varies from field to field. As such, we refer to the following word pairs interchangeably: (network, graph), (vertex, node), (edge, link). Throughout the paper, we follow standard practice and use capital bold letters for matrices (e.g., $\boldsymbol{A}$), lower-case bold letters for vectors (e.g., $\boldsymbol{a}$). Also, we focus on undirected and unweighted graphs.

2 TIGER ROBUSTNESS MEASURES

Tiger contains 22 robustness measures, grouped into one of three categories depending on whether the measure utilizes the graph, adjacency, or Laplacian matrix. In Table 1, we highlight each implemented measure, the category it belongs to (graph, adjacency, Laplacian), and its application to network robustness. Next, we select 3 robustness measures, one from each of the three categories, to extensively discuss. For detailed description and discussion of all 22 measures, we refer the reader to the online documentation.

2.1 Example Measures

Average vertex betweenness ($\bar{b}_v$) of a graph $G = (\mathcal{V}, \mathcal{E})$ is the summation of vertex betweenness b_u for every node $u \in V$, where vertex betweenness for node u is defined as the number of shortest paths that pass through u out of the total possible shortest paths

$$\bar{b}_v = \sum_{u \in V} \sum_{s \in V} \sum_{\substack{t \in V \\ s \neq t \neq u}} \frac{n_{s,t}(u)}{n_{s,t}} \tag{1}$$

where $n_{s,t}(u)$ is the number of shortest paths between s and t that pass through u and $n_{s,t}$ is the total number of shortest paths between s and t [12]. Average vertex betweenness has a natural connection to graph robustness since it measures the average load on vertices in the network. The smaller the average the more robust the network, since load is more evenly distributed across nodes.

Spectral scaling (ξ) indicates if a network is simultaneously sparse and highly connected, known as "good expansion" (GE) [11, 20]. Intuitively, we can think of a network with GE as a network lacking bridges or bottlenecks. In order to determine if a network has GE, [11] proposes to combine the spectral gap measure with odd subgraph centrality SC_{odd}, which measures the number of odd length closed walks a node participates in. Formally, *spectral scaling* is described in Equation 2,

$$\xi(G) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \{log[\boldsymbol{u}_1(i)] - [log\boldsymbol{A} + \frac{1}{2} log[SC_{odd}(i)]]\}^2} \tag{2}$$

where $\boldsymbol{A} = [sinh(\lambda_1)]^{-0.5}$, n is the number of nodes, and $\boldsymbol{u}_1$ is the first eigenvector of adjacency matrix $\boldsymbol{A}$. The closer ξ is to zero, the better the expansion properties and the more robust the network. Formally, a network is considered to have GE if $\xi < 10^{-2}$, the correlation coefficient $r < 0.999$ and the slope is 0.5.

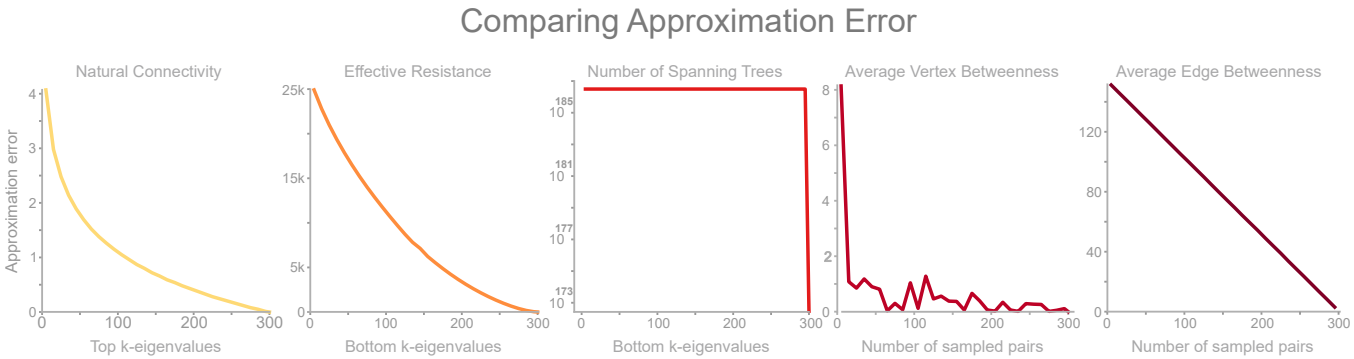

Figure 2: Error of 5 fast, approximate robustness measures supported by Tiger. Parameter k represents the trade-off between speed (low k) and precision (high k). To measure approximation efficacy, we vary $k \in [5, 300]$ in increments of 10 and measure the absolute error between the approximate and original measure averaged over 30 runs on a clustered scale-free graph with 300 nodes. [sf: need to say that this is relative error]

Effective resistance (R) views a graph as an electrical circuit where an edge (i, j) corresponds to a resister of $r_{ij} = 1$ Ohm and a node i corresponds to a junction. As such, the effective resistance between two vertices i and j, denoted R_{ij}, is the electrical resistance measured across i and j when calculated using Kirchoff's circuit laws. Extending this to the whole graph, we say the *effective graph resistance* R is the sum of resistances for all distinct pairs of vertices [9, 15]. Klein and Randic [23] proved this can be calculated based on the sum of the inverse non-zero Laplacian eigenvalues:

$$R = \frac{1}{2}\sum_{i,j}^{n} R_{ij} = n\sum_{i=2}^{n}\frac{1}{\mu_i} \tag{3}$$

As a robustness measure, effective resistance measures how well connected a network is, where a smaller value indicates a more robust network [9, 15]. In addition, the effective resistance has many desirable properties, including the fact that it strictly decreases when adding edges, and takes into account both the number of paths between node pairs and their length [10].

2.2 Measure Implementation & Evaluation

Our goal for Tiger is to implement each robustness measure in a clear and concise manner to facilitate code readability, while simultaneously optimizing for execution speed. Each robustness measure is wrapped in a function that abstracts mathematical details away from the user; and any default parameters are set for a balance of *speed* and *precision*. Below we compare the efficacy of 5 fast, approximate robustness measures, followed by an analysis of the scalability of all 22 measures.

Approximate Measures. It turns out that a large number of robustness measures have difficulty scaling to large graphs. To help address this, we implement and compare 5 fast approximate measure, three spectral based (natural connectivity, number of spanning trees, effective resistance), and two graph based (average vertex betweenness, average edge betweenness) [4, 5]. To approximate *natural connectivity* we use the top-k eigenvalues of the adjacency matrix as a low rank approximation [5, 28]. For the *number of spanning trees* and *effective resistance* we take the bottom-k eigenvalues of the Laplacian matrix [5]. For graph measures, *average vertex betweenness* and *average edge betweenness*, we randomly sample k nodes to calculate centrality. In both cases, the parameter k represents the trade-off between speed (low k) and precision (high k). When k is equal to the number of nodes n in the graph, the approximate measure is equivalent to the original.

To determine the efficacy of each approximation measure we vary $k \in [5, 300]$ in increments of 10, and measure the absolute error between the approximate and original measure, averaged over 30 runs on a clustered scale free graph containing 300 nodes. In Figure 2, we observe that average vertex betweenness accurately approximates the original measure using ~10% of the nodes in the graph. This results in a significant speed-up, and is in line with prior research [4]. While the absolute error for each spectral approximation is large, these approximations find utility in measuring the relative change in graph robustness after a series of perturbations (i.e., addition or removal of nodes/edges). While not immediately obvious, this can enable the development a wide range of optimization based defense techniques [5, 6].

Scalability. To help Tiger users determine which measures would scale to large graphs, we compare and report the scalability of all 22 robustness measures on 5 clustered scale-free graphs ranging from 100, 1k, 10k 100k, to 1M nodes, averaged over 10 random runs, as shown in Figure 3. We compute every measure on each of the 5 graphs for 30 minutes, and observe clear trends allowing us to assess the measures' potential to scale to larger graphs. For approximate edge and vertex betweenness measures we set $k = 0.1n$, where n is the number of nodes [4]. For spectral measure we set $k = 30$ [6]. We find that 7 measures are highly scalable, i.e., running in less than 30 minutes on a 1M million node graph, with 2 notable runner-ups that are able to run on a 100k graph in less than 30 minutes. We consider the remaining measures to scale poorly.

3 TIGER ATTACKS

There are two primary ways a network can become damaged—(1) *natural failure* and (2) *targeted attack*. Natural failures typically occur when a piece of equipment breaks down from natural causes. In the study of graphs, this would correspond to the removal of a

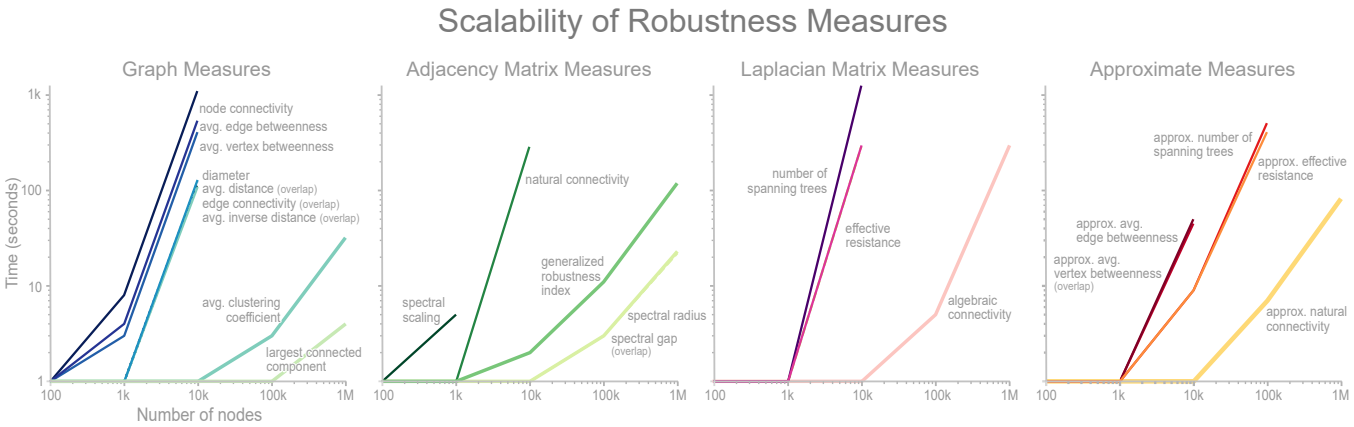

Figure 3: Scalability of TIGER's 22 robustness measures on a clustered scale free graph ranging from 100, 1k, 10k 100k, to 1M nodes averaged over 10 runs. We find 7 measures are highly scalable i.e., runs in <30 minutes on a 1M node graph.

node or edge in the graph. While random network failures regularly occur, they are typically less severe than targeted attacks. This has been shown to be true across a range of graph structures [3, 33]. In contrast, targeted attacks carefully select nodes and edges in the network for removal in order to maximally disrupt network functionality. As such, we focus the majority of our attention to targeted attacks. In Section 3.1, we provide a high-level overview of several network failure and attack strategies. Then, in Section 3.2 we highlight 10 attack strategies implemented in TIGER.

3.1 Attack Strategies

We showcase an example attack in Figure 5 on the Kentucky KY-2 water distribution network [17]. The network starts under normal conditions (far left), and at each step an additional node is removed by the attacker (red nodes). After removing only 13 of the 814 nodes, the network is split into two separate regions. By step 27, the network splits into four disconnected regions. In this simulation, and in general, attack strategies rely on node and edge centrality measures to identify candidates. Below, we highlight several attack strategies [19] contained in TIGER.

Initial degree removal (ID) targets nodes with the highest degree δ_v. This has the effect of reducing the total number of edges in the network as fast as possible [19]. Since this attack only considers its neighbors when making a decision, it is considered a *local attack*. The benefit of this locality is low computational overhead.

Initial betweenness removal (IB) targets nodes with high betweenness centrality b_v. This has the effect of destroying as many paths as possible [19]. Since path information is aggregated from across the network, this is considered a *global attack* strategy. Unfortunately, global information comes with significant computational overhead compared to a local attacks.

Recalculated degree (*RD*) and **betweenness removal (*RB*)** follow the same process as *ID* and *IB*, respectively, with one additional step to recalculate the degree (or betweenness) distribution after a node is removed. This recalculation often results in a stronger attack, however, recalculating these distributions adds a significant amount of computational overhead to the attack.

3.2 Comparing Strategies

To help TIGER users determine the effectiveness of attack strategies, we evaluate 5 node and 5 edge attacks on the Kentucky KY-2 water distribution network in Figure 4. We begin by analyzing each node attack strategy—*ID*, *RD*, *IB*, *RB* and *RND* (random selection)—on the left-side of Figure 4. Attack success is measured based on how fractured the network becomes when removing nodes from the network. We identify three key observations—(i) random node removal (*RND*) is not an effective strategy on this network structure; (ii) *RB* is the most effective attack strategy; and (iii) the remaining three attacks are roughly equivalent, falling somewhere between *RND* and *RB*.

Analyzing Figure 5, we can gain insight into why *RB* is the most effective of the attacks. If we look carefully, we observe that certain nodes (and edges) in the network act as key bridges between various network regions. As a result, attacks able to identify these bridges are highly effective in disrupting this network. In contrast, degree based attacks are less effective, likely due to the balanced degree distribution. The analysis is similar for edge based attacks.

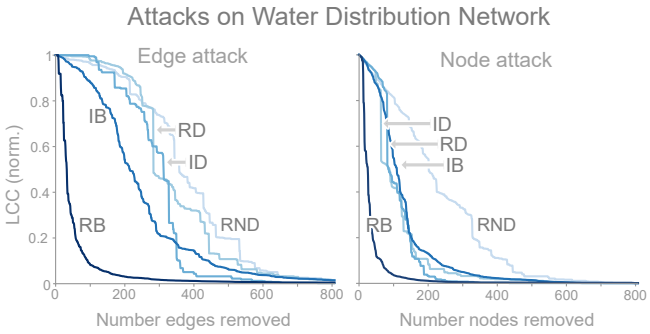

Figure 4: Efficacy of 5 edge attacks (left) and 5 node attacks (right) on the KY-2 water distribution network. The most effective attack (RB) disconnects approximately 50% of the network with less than 30 removed edges (or nodes).

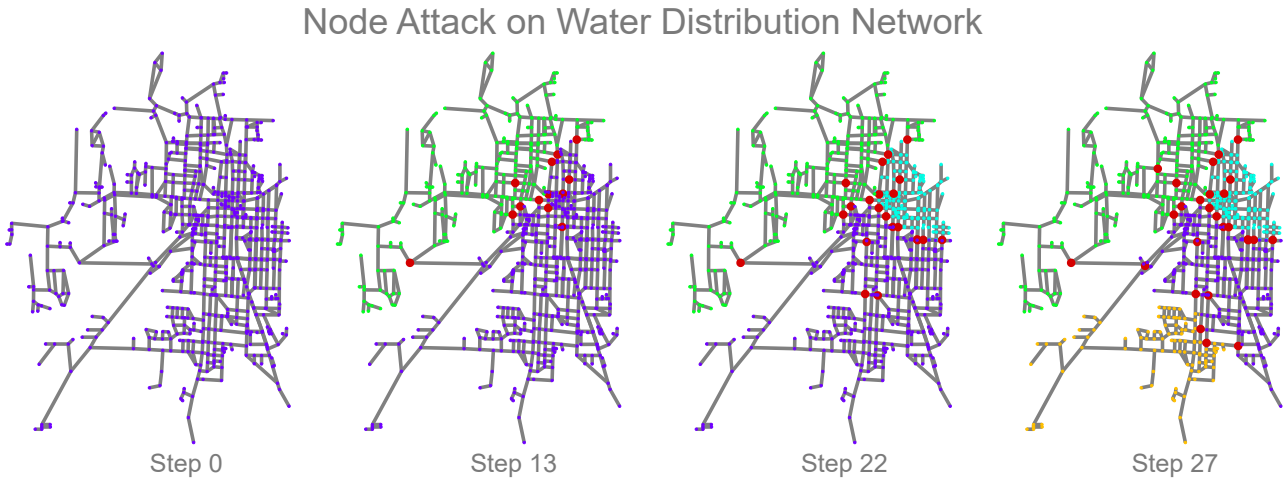

Figure 5: Tiger simulation of an RD node attack on the KY-2 water distribution network. Step 0: network starts under normal conditions; at each step a node is removed by the attacker (red nodes). Step 13, 22 & 27: after removing only a few of the 814 nodes, the network splits into two and three and four disconnected regions, respectively.

4 TIGER DEFENSES

The same centrality measures effective in attacking a network are important to network defense (e.g., degree, betweenness, PageRank, eigenvector, etc.). In fact, if an attack strategy is known a priori, node monitoring can largely prevent an attack altogether. In Section 4.1, we provide a high-level overview of several heuristic and optimization based defense techniques. Then, in Section 4.2 we show Tiger users how several defense techniques can be used to robustify an attacked network.

4.1 Defense Strategies

We categorize defense techniques based on whether they operate heuristically, modifying graph structure independent of a robustness measure, or by optimizing for a particular robustness measure [5]. Within each category a network can be defended i.e., improve its robustness by—(1) *edge rewiring*, (2) *edge addition*, or (iii) *node monitoring*. Edge rewiring is considered a *low* cost, *less* effective version of edge addition. On the other hand, edge addition almost always provides stronger defense [3]. Node monitoring provides an orthogonal mechanism to increase network robustness by monitoring (or removing) nodes in the graph. This has an array of applications, including: (i) preventing targeted attacks, (ii) mitigating cascading failures, and (iii) reducing the spread of network entities. Below, we highlight several heuristic and optimization based techniques contained in Tiger.

Heuristic Defenses. We overview 5 edge rewiring and addition defenses [3], and compare the effectiveness of them in Section 4.2:

1. *Random addition*: adds an edge between two random nodes.
2. *Preferential addition*: adds an edge connecting two nodes with the lowest degrees.
3. *Random edge rewiring*: removes a random edge and adds one using (1).
4. *Random neighbor rewiring*: randomly selects neighbor of a node and removes the edge. An edge is then added using (1).
5. *Preferential random edge rewiring*: selects an edge, disconnects the higher degree node, and reconnects to a random one.

Optimization Defenses. We discuss the Netshield node monitoring technique which identifies key nodes in a network to reduce the spread of entity dissemination (e.g., viruses) [30]. To minimize the spread of entities, Netshield minimizes the spectral radius of the graph λ_1 by selecting the best set S of k nodes to remove from the graph (i.e., maximize eigendrop). In order to evaluate the goodness of a node set S for removal, [30] proposes the Shield-value measure:

$$Sv(S) = \sum_{i \in S} 2\lambda_1 \boldsymbol{u}_1(i)^2 - \sum_{i,j \in S} A(i,j)\boldsymbol{u}(i)\boldsymbol{u}(j) \tag{4}$$

The intuition behind this equation is to select nodes for monitoring that have high eigenvector centrality (first term), while penalizing neighboring nodes to prevent grouping (second term). We demonstrate the utility of this defense mechanism in Section 5.

4.2 Comparing Strategies

To help users evaluate the effectiveness of defense techniques, we compare 5 edge defenses on the Kentucky KY-2 water distribution network, averaged over 10 runs, in Figure 6. The network is initially attacked using the *RB* attack strategy (30 nodes removed), and the success of each defense is measured based on how it can reconnect the network by adding or rewiring edges in the network (higher is better). Based on Figure 6, we identify three key observations—(i) preferential edge addition performs the best; (ii) edge addition, in general, outperforms rewiring strategies; and (iii) random neighbor rewiring typically performs better than the other rewiring strategies.

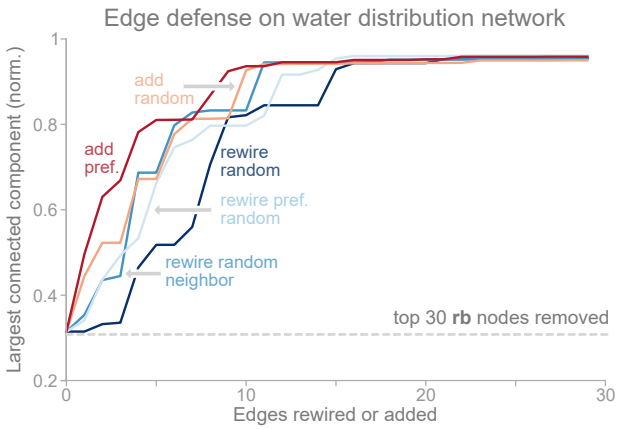

Figure 6: Comparing ability of 5 edge defenses to improve KY-2 network robustness after removing 30 nodes via RB attack. Edge addition performs the best, with random edge rewiring performing the worst.

5 TIGER SIMULATION TOOLS

We implement 4 broad and important types of robustness simulation tools [3, 19, 22, 30, 32]—(1) dissemination of network entities, (2) cascading failures (3) network attacks, see Section 3, and (4) network defense, see Section 4. In Section 5.1, we discuss the implementation of an infectious disease models and how defense techniques implemented in Tiger can be used to either *minimize* or *maximize* the network diffusion. Then, in Section 3, we discuss the implementation of the cascading failure model and its interactions with Tiger defense and attack strategies.

5.1 Dissemination of Network Entities

A critical concept in entity dissemination is *network diffusion*, which attempts to capture the underlying mechanism enabling network propagation. In order to augment this diffusion process, Tiger leverages the defense techniques in Section 4 for use with two prominent diffusion models: the flu-like susceptible-infected-susceptible (SIS) model, and the vaccinated-like susceptible-infected-recovered (SIR) model [22]. For example, to *minimize* the ability of viruses to spread we can monitor (remove) nodes in the graph to reduce graph connectivity. On the other hand, if want to *maximize* network diffusion e.g., marketing campaign, we can use defense techniques like edge rewiring or addition to increase graph connectivity. Below, we highlight the SIS infectious disease model and how Tiger's defense techniques can help contain a simulated outbreak.

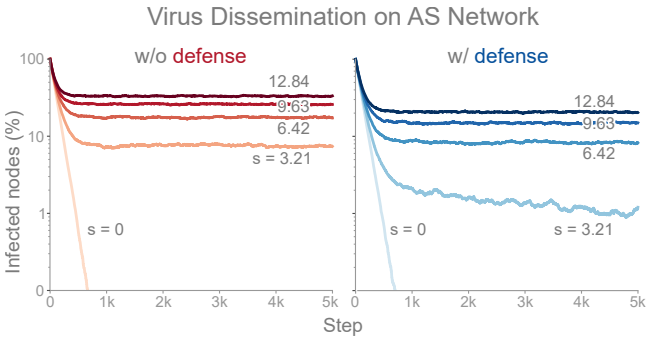

Figure 7: SIS simulation with 5 virus strengths on the Oregon-1 Autonomous System network. No defense (left), Netshield defense (right).

Design and Implementation. Each node in the SIS model can be in one of two states, infected I or susceptible S. At each time step t, an infected node v has a probability β of infecting each of it's uninfected neighbors $u \in N(v)$. After this, each infected node v has a probability δ of healing and becoming susceptible again. The relationship between the birth rate β, death rate δ and the spectral radius λ_1 of the graph has been a widely studied topic. In [31], they show that the spectral radius of a graph is closely tied to the epidemic threshold τ of a network in an SIS model. In particular, they prove that $\frac{\beta}{\delta} < \tau = \frac{1}{\lambda_1}$. This means for a given virus strength s, an epidemic is more likely to occur on a graph with larger λ_1. As such, we say that a virus has an effective strength $s = \lambda_1 \cdot b/d$, where a larger s means a stronger virus [30].

Simulating dissemination of entities. To help users visualize the dissemination process, we enable them to create visuals like Figure 1, where we run an SIS computer virus simulation ($s = 3.21$) on the Oregon-1 Autonomous System network [27]. The top of Figure 1 shows the virus progression when defending 5 nodes selected by Netshield [30]. By time step 1000, the virus has nearly died out. The bottom of Figure 1 shows that the virus remains endemic without defense. To assist users in summarizing model results over many configurations, we enable them to create plots like Figure 7, which show results for 5 SIS effective virus strengths $s = \{0, 3.21, 6.42, 9.63, 12.84\}$ over a period of 5000 steps.

5.2 Cascading Failures

Cascading failures often arise as a result of natural failures or targeted attacks in a network. Consider an electrical grid where a central substation goes offline. In order to maintain the distribution of power, neighboring substations have to increase production in order to meet demand. However, if this is not possible, the neighboring substation fails, which in turn causes additional neighboring substations to fail. The end result is a series of cascading failures i.e., a blackout [8]. While cascading failures can occur in a variety of network types e.g., water, electrical, communication, we focus on the electrical grid. Below, we discuss the design and implementation of the cascading failure model and how Tiger can be used to both *induce* and *prevent* cascading failures using the attack and defense mechanisms discussed in Sections 3 and 4, respectively.

Design and Implementation. There are 3 main processes governing the network simulation—(1) *capacity* of each node $c_v \in [0, 1]$; (2) *load* of each node $l_v \in U(0, l_{max})$; and (3) network *redundancy* $r \in [0, 1]$. The capacity of each node c_v is the the maximum load a node can handle, which is set based on the node's normalized betweenness centrality [18]. The load of each node l_v represents the fraction of maximum capacity c_v that the node operates at. Load for each node c_v is set by uniformly drawing from $U(0, l_{max})$, where l_{max} is the maximum initial load. Network redundancy r represents the amount of reserve capacity present in the network i.e., auxiliary

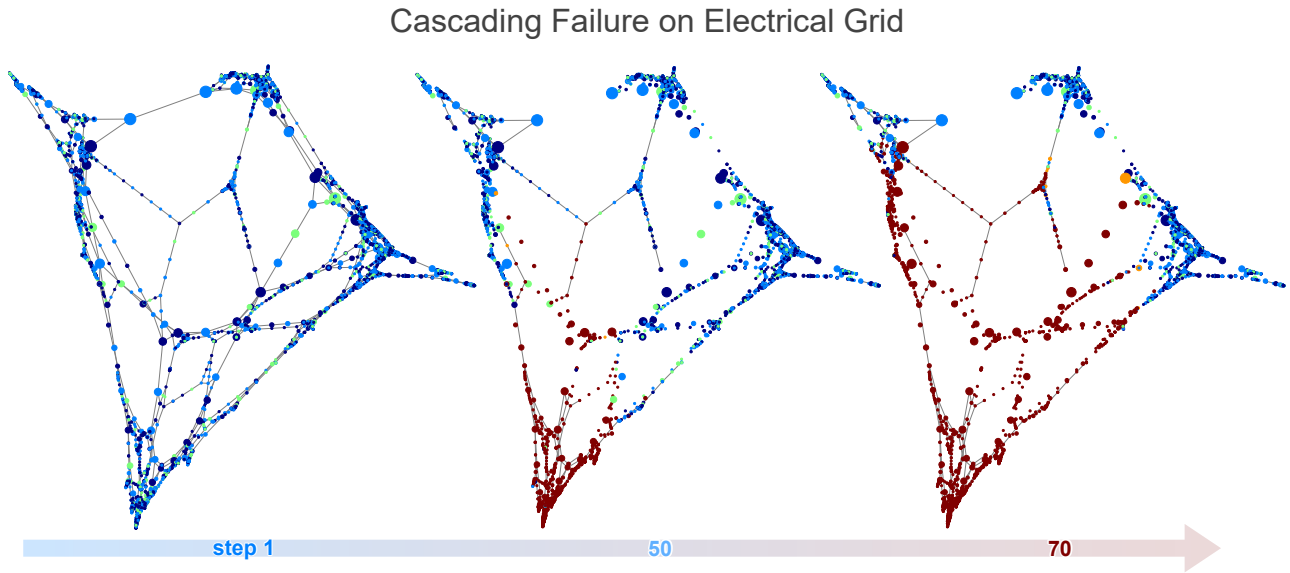

Figure 8: Tiger cascading failure simulation on the US power grid network when 4 nodes are overloaded according to the ID attack strategy. Time step 1: shows the network under normal conditions. Time step 50: we observe a series of failures originating from the bottom of the network. Time step 70: most of the network has collapsed.

support systems. At the beginning of the simulation, we allow the user to attack and defend the network according to the node attack and defense strategies in Sections 3 and 4, respectively. When a node is attacked it becomes "overloaded", causing it to fail and requiring the load be distributed to the neighbors. When defending a node we increase it's capacity to protect against attacks.

Simulating cascading failures. To help users visualize cascading failures induced by targeted attacks, we enable them to create visuals like Figure 8, where we overload 4 nodes selected by the ID attack strategy on the US power grid dataset [32] ($l_{max} = 0.8$). Node size represents capacity i.e., larger size $\rightarrow$ higher capacity, and color indicates the load of each node on a gradient scale from blue (low load) to red (high load); dark red indicates node failure (overloaded). Time step 1 shows the network under normal conditions; at step 50 we observe a series of failures originating from the bottom of the network; by step 70 most of the network has collapsed. To assist users in summarizing simulation results over many configurations, we enable them to create plots like Figure 9, which shows the effect of network redundancy r when 4 nodes are overloaded by the ID attack strategy. At 50% redundancy, we observe a critical threshold where the network is able to redistribute the increased load. For $r < 50\%$, the cascading failure can be delayed but not prevented.

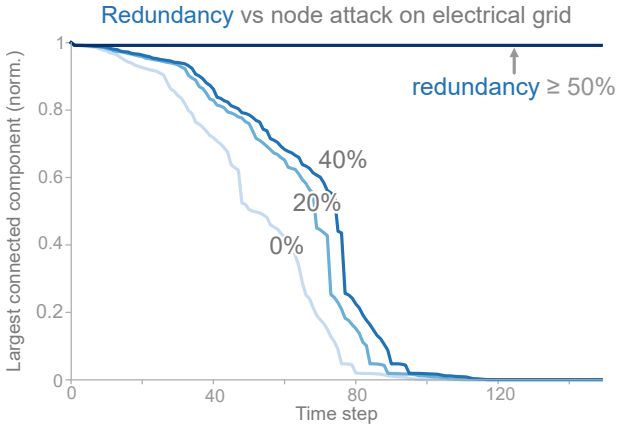

Figure 9: Effect of network redundancy r on the US power grid where 4 nodes are overloaded using ID. When $r \geq 50\%$ the network is able to redistribute the increased load. [sf: change gradient bar to represent additional color availability (green, yellow, orange, red)]

6 CONCLUSION

The study of network robustness is a critical tool in the characterization and understanding of complex interconnected systems. Through analyzing and understanding the robustness of these networks we can: (1) quantify network vulnerability and robustness, (2) augment a network's structure to resist attacks and recover from failure, and (3) control the dissemination of entities on the network (e.g., viruses, propaganda). While significant research has been conducted on all of these tasks, no comprehensive open-source toolbox currently exists to assist researchers and practitioners in this important topic. This lack of available tools hinders reproducibility and examination of existing work, development of new research, and dissemination of new ideas. To address these challenges, we contribute Tiger, an open-sourced Python toolbox containing 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization based defense techniques; and 4 simulation tools. Tiger is open-sourced at: https://github.com/safreita1/TIGER.

7 ACKNOWLEDGEMENTS

This work was in part supported by the NSF grant IIS-1563816, GRFP (DGE-1650044), and a Raytheon fellowship.

REFERENCES

[1] Asma Azizi, Cesar Montalvo, Baltazar Espinoza, Yun Kang, and Carlos Castillo-Chavez. 2020. Epidemics on networks: Reducing disease transmission using health emergency declarations and peer communication. *Infectious Disease Modelling* 5 (2020), 12–22.

[2] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: an open source software for exploring and manipulating networks. In *Third international AAAI conference on weblogs and social media*.

[3] Alina Beygelzimer, Geoffrey Grinstein, Ralph Linsker, and Irina Rish. 2005. Improving network robustness by edge modification. *Physica A: Statistical Mechanics and its Applications* 357, 3-4 (2005), 593–612.

[4] Ulrik Brandes and Christian Pich. 2007. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos* 17, 07 (2007), 2303–2318.

[5] Hau Chan and Leman Akoglu. 2016. Optimizing network robustness by edge rewiring: a general framework. *Data Mining and Knowledge Discovery* 30, 5 (2016), 1395–1425.

[6] Hau Chan, Leman Akoglu, and Hanghang Tong. 2014. Make it or break it: Manipulating robustness in large networks. In *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 325–333.

[7] Vasek Chvátal. 1973. Tough graphs and Hamiltonian circuits. *Discrete Mathematics* 5, 3 (1973), 215–228.

[8] Paolo Crucitti, Vito Latora, and Massimo Marchiori. 2004. Model for cascading failures in complex networks. *Physical Review E* 69, 4 (2004), 045104.

[9] Wendy Ellens and Robert E Kooij. 2013. Graph measures and network robustness. *arXiv preprint arXiv:1311.5064* (2013).

[10] Wendy Ellens, FM Spieksma, P Van Mieghem, A Jamakovic, and RE Kooij. 2011. Effective graph resistance. *Linear algebra and its applications* 435, 10 (2011), 2491–2506.

[11] Ernesto Estrada. 2006. Network robustness to targeted attacks. The interplay of expansibility and degree distribution. *The European Physical Journal B-Condensed Matter and Complex Systems* 52, 4 (2006), 563–574.

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

[13] Scott Freitas, Hanghang Tong, Nan Cao, and Yinglong Xia. 2017. Rapid analysis of network connectivity. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2463–2466.

[14] Scott Freitas, Andrew Wicker, Duen Horng Chau, and Joshua Neil. 2020. D2M: Dynamic Defense and Modeling of Adversarial Movement in Networks. *SDM* (2020).

[15] Arpita Ghosh, Stephen Boyd, and Amin Saberi. 2008. Minimizing effective resistance of a graph. *SIAM review* 50, 1 (2008), 37–66.

[16] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[17] Erika Hernadez, Steven Hoagland, and Lindell Ormsbee. 2016. Water distribution database for research applications. In *World Environmental and Water Resources Congress 2016*. 465–474.

[18] Isaac Hernandez-Fajardo and Leonardo Dueñas-Osorio. 2013. Probabilistic study of cascading failures in complex interdependent lifeline systems. *Reliability Engineering & System Safety* 111 (2013), 260–272.

[19] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. 2002. Attack vulnerability of complex networks. *Physical review E* 65, 5 (2002), 056109.

[20] Shlomo Hoory, Nathan Linial, and Avi Wigderson. 2006. Expander graphs and their applications. *Bull. Amer. Math. Soc.* 43, 4 (2006), 439–561.

[21] Jian Kang, Scott Freitas, Haichao Yu, Yinglong Xia, Nan Cao, and Hanghang Tong. 2018. X-rank: Explainable ranking in complex multi-layered networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1959–1962.

[22] William Ogilvy Kermack and Anderson G McKendrick. 1927. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character* 115, 772 (1927), 700–721.

[23] Douglas J Klein and Milan Randić. 1993. Resistance distance. *Journal of mathematical chemistry* 12, 1 (1993), 81–95.

[24] Katherine A Klise, Regan Murray, and Terra Haxton. 2018. *An Overview of the Water Network Tool for Resilience (WNTR)*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[25] Mert Korkali, Jason G Veneman, Brian F Tivnan, James P Bagrow, and Paul DH Hines. 2017. Reducing cascading failure risk by increasing infrastructure network interdependence. *Scientific reports* 7 (2017), 44499.

[26] Mukkai S Krishnamoorthy and Balaji Krishnamurthy. 1987. Fault diameter of interconnection networks. *Computers & Mathematics with Applications* 13, 5-6 (1987), 577–582.

[27] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.

[28] Fragkiskos D Malliaros, Vasileios Megalooikonomou, and Christos Faloutsos. 2012. Fast robustness estimation in large social graphs: Communities and anomaly detection. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 942–953.

[29] Giulio Rossetti, Letizia Milli, Salvatore Rinzivillo, Alina Sîrbu, Dino Pedreschi, and Fosca Giannotti. 2018. NDlib: a python library to model and analyze diffusion processes over complex networks. *International Journal of Data Science and Analytics* 5, 1 (2018), 61–79.

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

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

[32] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440.

[33] Yongxiang Xia, Jin Fan, and David Hill. 2010. Cascading failure in Watts–Strogatz small-world networks. *Physica A: Statistical Mechanics and its Applications* 389, 6 (2010), 1281–1285.