APPLICATION OF GROUP TESTING FOR ANALYZING NOISY NETWORKS

Vladimir Ufimtsev
University of Nebraska at Omaha

Follow this and additional works at: https://digitalcommons.unomaha.edu/studentwork

Part of the Computer Sciences Commons

Please take our feedback survey at: https://unomaha.az1.qualtrics.com/jfe/form/SV_8cchtFmpDyGfBLE

Recommended Citation
https://digitalcommons.unomaha.edu/studentwork/2909

This Dissertation is brought to you for free and open access by DigitalCommons@UNO. It has been accepted for inclusion in Student Work by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.
APPLICATION OF GROUP TESTING FOR ANALYZING NOISY NETWORKS

By

Vladimir Ufimtsev

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Information Technology

Under the Supervision of Dr. Sanjukta Bhowmick

Omaha, Nebraska

July, 2016

Supervisory Committee

Dr. Hesham Ali

Dr. Prithviraj Dasgupta

Dr. Lotfollah Najjar

Dr. Vyacheslav Rykov
Abstract

Application of Group Testing for Analyzing Noisy Networks

Vladimir Ufimtsev, Ph.D.

University of Nebraska, 2016

Advisor: Dr. Sanjukta Bhowmick

My dissertation focuses on developing scalable algorithms for analyzing large complex networks and evaluating how the results alter with changes to the network. Network analysis has become a ubiquitous and very effective tool in big data analysis, particularly for understanding the mechanisms of complex systems that arise in diverse disciplines such as cybersecurity [83], biology [15], sociology [5], and epidemiology [7]. However, data from real-world systems are inherently noisy because they are influenced by fluctuations in experiments, subjective interpretation of data, and limitation of computing resources. Therefore, the corresponding networks are also approximate. This research addresses these issues of obtaining accurate results from large noisy networks efficiently.

My dissertation has four main components. The first component consists of developing efficient and scalable algorithms for centrality computations that produce reliable results on noisy networks. Two novel contributions I made in this area are the development of a group testing [16] based algorithm for identification of high centrality vertices which is extremely faster than current methods, and an algorithm for computing the betweenness centrality of a specific vertex.

The second component consists of developing quantitative metrics to measure how different noise models affect the analysis results. We implemented a uniform perturbation model based on random addition/deletion of edges of a network. To quantify the stability of a network we investigated the effect that perturbations have on the top-k ranked vertices and the local structure properties of the top ranked vertices.

The third component consists of developing efficient software for network analysis. I have been part of the development of a software package, ESSENS (Extensible, Scalable Software for Evolving NetworkS) [76], that effectively supports our algorithms on large networks.

The fourth component is a literature review of the various noise models that researchers have applied to networks and the methods they have used to quantify the stability, sensitivity, robustness, and reliability of networks.

These four aspects together will lead to efficient, accurate, and highly scalable algorithms for analyzing noisy networks.
Author’s Acknowledgements

I would like to express deep gratitude to my advisor, Dr. Sanjukta Bhowmick, for all of her insight, support, and guidance throughout my years as a Ph.D. student. Your insight and suggestions have been instrumental in guiding my dissertation. Thank you for being my advisor and keeping me on the right track.

I would also like to express special thanks to my undergraduate advisor, who is also a member of my doctoral supervisory committee, Dr. Vyacheslav Rykov. You are the person who introduced me to research in science and mathematics, and provided me with a solid foundation from which I could work. For this I am extremely grateful.

Thank you to all of the members of my supervisory committee: Dr. Sanjukta Bhowmick, Dr. Vyacheslav Rykov, Dr. Hesham Ali, Dr. Prithviraj Dasgupta, and Dr. Lotfollah Najjar. I really appreciate all of the suggestions made during my proposal and dissertation defense for making my work better and what I could pursue in the future. Thank you for all of your effort in serving on my committee.

I would like to thank all of our collaborators including: A. Vullikanti and A. Adiga for their support with the perturbations model, A. Mukherjee and S. Sarkar for helping us with our recent work on noise in networks, K. Cooper (Dempsey), the network science research group at UNO, and the bioinformatics research group at UNO.

Finally, I would like to thank my family for all of their support and patience during my whole student career.

Grant Acknowledgement Information

This work was supported by: GRACA (UNO), F.I.R.E. (UNO Sponsored Programs), College of IS&T (UNO), and NSF-CCF Award # 153881.
## Contents

1 Introduction 1

1.1 Identifying Important Vertices in Networks ............................... 1
  1.1.1 Group Testing ........................................ 1
  1.1.2 Parallel Implementation ................................. 3

1.2 Network Noise .................................................. 3
  1.2.1 Centrality Measures Analyzed .......................... 3
  1.2.2 Vertex Ranking ......................................... 4

1.3 Network Analysis Software .................................... 4

1.4 Main Contributions ............................................ 4

2 Background 6

2.1 Graph Theory .................................................. 6

2.2 Centrality Measures ......................................... 6

2.3 Group Testing .................................................. 7
  2.3.1 Simple Group Testing Example ........................ 7
  2.3.2 Previous Work on Group Testing on Graphs ......... 8

2.4 Network Noise Model ......................................... 9

2.5 Parallel Graph Algorithms ................................ 10

2.6 Existing Network Analysis Software ........................ 10

2.7 Input Data ..................................................... 11
  2.7.1 Real-World Networks .................................. 11
  2.7.2 Random Networks - RMAT ............................ 12
  2.7.3 Random Networks - LFR ............................... 12

3 Group Testing on Networks 13

3.1 Method ......................................................... 13
  3.1.1 Theoretical Background ................................ 14
  3.1.2 Latin Square Superimposed Code Construction .... 15

3.2 Betweenness Centrality Results ............................ 19
  3.2.1 Scalability Results ................................... 19
  3.2.2 Algorithm for Betweenness Centrality of a Single Vertex 20

3.3 Closeness Centrality Results ............................... 22
A  Other Group Testing Designs  66
A.1  Construction of a Superimposed Code Using Partitions 66
A.1.1  Partial Order on Set Partitions 66
A.1.2  Superimposed Code Construction 67
A.2  Superimposed Codes Analysis for Multiple Access Channels and Group Testing 69
A.2.1  Multiple Access Channel 69
A.2.2  ALOHA System 69
A.2.3  Mathematical Modeling of the MAC-OR and Slotted ALOHA System 70
A.2.4  MAC-OR Channel Model 71
A.2.5  Slotted Aloha System Model 71
A.2.6  MDS Codes 73
A.2.7  Superimposed Codes from Reed-Solomon Codes 74
A.2.8  Slotted ALOHA System Performance 75
A.2.9  Characteristics of the System 75
A.2.10  Previous Lower Bounds on Channel Capacity 76
A.2.11  Improving the Lower Bound on Channel Capacity 76
A.2.12  Lower Bound on Channel Capacity of MAC-OR in the Slotted ALOHA System 78
A.2.13  Application to Group Testing 81

B  Supplementary Results in Network Noise Analysis  82
B.1  Additional Noise Charts 82
List of Figures

1. Basic Outline of a Screening Experiment Using Group Testing. .......................... 2
2. Finding 1 Defective Unit From Population With $N = 8$ Units. ............................ 8
3. Example of Group Testing for BC Using Latin Squares. ................................. 16
4. Example of Group Testing for CC Using Latin Squares. ................................. 17
5. Illustration of Parallel Process. ................................................................. 18
6. Strong Scalability Results of Group Testing. ................................................ 21
7. Ratio of the Execution Time of the Parallel Closeness Centrality Algorithms to the Parallel Group Testing Algorithm. .............................................................. 25
8. Comparison of the Running Time for the Parallel Closeness Centrality Algorithms and the Parallel Group Testing Algorithm on Two Cores. .................................................. 25
10. Effect of perturbation on the ranking of the high BC vertices. .......................... 36
11. Effect of perturbation on the ranking of the high CC vertices. .......................... 37
12. Changes in dominant stability of centrality metrics over different noise levels. ..... 39
13. Change in centrality values for different networks and various choices of $k$, over different noise levels. ................................................................. 41
14. Stable clusters of centrality values ................................................................. 42
15. High ranked common neighbors of the top-$k$ ranked vertices. ........................ 45
16. Subgraphs for high betweenness (top) and high closeness (bottom) centrality vertices. ................................................................. 46
17. Stability of centrality metrics for synthetic networks. ....................................... 48
18. Blueprint of the ESSENS Framework Showing the Levels of Abstraction. ........ 52
19. Comparison of Runtime for Betweenness Centrality using the Brandes Method of ESSENS with Boost Graph Library. .................................................. 52
20. Karate Network Perturbations. ................................................................. 82
21. Chesapeake Network Perturbations. ........................................................... 83
22. LesMis Network Perturbations. ................................................................. 83
23. Celegans Network Perturbations. ............................................................... 84
24. AS1 Network Perturbations. ................................................................. 84
25. AS2 Network Perturbations. ................................................................. 85
26. GrQc Network Perturbations. ................................................................. 85
27. HepTh Network Perturbations. ............................................................... 86
## List of Tables

1. Test Suite of Real-World Networks. .................................................. 12  
2. Finding High BC Vertices Using Group Testing on Real-World Networks. .... 20  
3. Test Suite of Networks for Closeness Centrality. ................................... 23  
5. Execution Time (sec) of The Networks. .............................................. 24  
6. Standard deviations for changes to the ranking of the top 10 BC vertices .......... 36  
7. Test suite of real-world networks and their properties. ........................... 39  
8. Comparing Stability and Local Connections of Networks. ........................ 47
1 Introduction

Systems of interacting entities from diverse disciplines, such as information technology, biology, sociology, and epidemiology, can be modeled as networks [81], [7], [5]. Networks (also termed as graphs) consist of a set of vertices that correspond to the entities in the system, and a set of edges that correspond to the interaction between a pair of entities. Analyzing the properties of networks can help us understand the characteristics of the underlying system. Therefore, in recent years, many network analysis algorithms have been developed for a variety of different purposes such as efficiently identifying communities in networks and computing centrality metrics to measure the importance of the vertices [55]. Another important research area is determining the stability of networks subjected to noise by analyzing how their centrality rankings are affected.

1.1 Identifying Important Vertices in Networks

One important objective in network analysis is to identify key vertices in the system, for example the most prominent people in a social network or lethal genes in a biological network. In network terms this translates to identifying high centrality vertices. Once the vertices are ranked by their centrality values (such as closeness centrality, betweenness centrality, degree centrality, and eigenvector centrality), then in most cases the high valued vertices represent the important entities. For most analysis purposes, only the high ranked vertices are required. However, most algorithms calculate the centrality values of all the vertices and then sort them to find the highest vertices [31], [9]. This process is inefficient because we only need to identify the significantly high vertices, and not their actual values. To achieve this goal more efficiently, we present an extremely fast and scalable algorithm for identifying the high ranked vertices, as per closeness centrality and betweenness centrality, using group testing.

1.1.1 Group Testing

Group testing [16], [48], [49], is a mathematical technique for finding a specified number of defective units among a large population of units. In a given population, units are termed as "defective" if they have a characteristic that is not present in the the other "non-defective" units. In group testing, the items are combined into carefully selected groups. For a given group and a given threshold, the presence (positive test) or absence (negative test) of the defective characteristic can be established by exactly one test. After \( N \) tests on a sufficient number of groups, we can exactly identify the defective units. \( N \) can be significantly smaller than the size of the population (which is how many tests you would have to do if testing each unit individually). Figure 1 provides a basic flowchart overview of the process. An important research topic is
designing the composition of the groups, such that the fewest number of tests are required to find the defective units. The best designs for group testing are known as superimposed codes. It has been shown ([20], [22], [19]) that asymptotically the number of required tests using superimposed codes is logarithmic in the total number of units.

The central idea in our application of group testing to networks is that when high centrality vertices are part of a group, and that group is combined into a supervertex, then the centrality of the supervertex should be high as well. Thus, when the centrality of the supervertex is above threshold $\tau$, then the result of the test will be positive (indicating presence of high centrality vertices), otherwise the result will be negative. Theoretically, by using an appropriate group testing design, it is possible to identify the highest centrality vertices.

My initial idea was to use group testing to identify high betweenness centrality vertices in networks, and we submitted a poster illustrating the idea and some preliminary results to the ACM Student Research Competition held at Supercomputing 2011 (SC11) [77]. The poster/presentation was awarded first place.

To date, there has been very limited implementation of group testing in networks. Examples include finding broken links in optical networks [33] and congested links in wireless sensor networks (WSNs) [11]. To the best of our knowledge, this is the first application of group testing to identifying sensitive vertices in complex networks. There is, however, previous work on identifying the vertices in a network with the highest closeness centrality. For example, in [57] ranking and approximation algorithms are used to obtain the ranking of the $k$ highest closeness centrality vertices in a network. These methods widely differ from

Figure 1: Basic Outline of a Screening Experiment Using Group Testing.
our group testing approach though the end goal is very similar to ours i.e. identifying important vertices in a network. More details and examples of group testing are presented in sections 2 and 3.

1.1.2 Parallel Implementation

Since we are developing the algorithms to be highly scalable and efficient enough to run on large networks, we have developed parallel versions of the group testing algorithm for both centrality metrics we focus on. To test the parallel implementations of the algorithms, we were able to utilize the parallel computing facilities at the Holland Computing Center at UNO. The technical specifications of the machines we used as well as the parallel implementation results are presented later in the results section.

1.2 Network Noise

A major issue in network analysis is that measurements of real-world systems are influenced by fluctuations in experimental conditions, subjective interpretation of data, and limitation of computing resources. Therefore the corresponding models are approximate. Network models are no exception. However, network analysis algorithms are typically based on graph theory, which traditionally dealt with exact inputs, and we have very little information on how combinatorial methods behave under inexact data. One focus of my research is to analyze how perturbations (noise in the network) affect network parameters. Specifically, we study the stability of centrality rankings of networks subjected to various levels of perturbation.

1.2.1 Centrality Measures Analyzed

The centrality measures that we primarily focus on are betweenness (BC) centrality and closeness centrality (CC) [55]. CC reflects how close a vertex is to all of the other vertices in the networks. For a given vertex \( v \), the closeness centrality of \( v \) is calculated by taking the reciprocal of the sum of the distance to all other vertices in the network. Vertices with the highest CC values are closer overall to the other vertices and are thus more central to the network. BC on the other hand measures how central a vertex is in the network in terms of information flow i.e. it calculates fractions of shortest paths between other vertices a given vertex is part of. Precise definitions of BC and CC are given in the next section. We have also performed analyses using the degree centrality measure. The degree of a given vertex is the number of direct neighbors it has i.e. the number of vertices directly connected to it. This measure indicates how important a particular vertex is "locally" in its neighborhood.
1.2.2 Vertex Ranking

The ranking of vertices, in the order of their importance for the different centrality measures, can change as the interactions in the system evolve. Ideally, the change in the vertex ranking should be proportional to the amount of change in the network. If there is a significant change, then the ranking should change significantly. Similarly, for a small change, the ranking should preferably remain about the same. This sensitivity to change is a crucial property, because real-world systems will incur some amount of experimental noise. This noise is generally manifested through small changes in the system. If such a small change completely alters the vertex ranking, then the measurements of importance will become meaningless. Therefore, as we identify the important vertices, we also need to ensure that these values will remain stable under noise.

1.3 Network Analysis Software

Another aspect of my work is assisting in the development of network analysis software. As we have been developing the algorithms, we have also been developing a software package, ESSENS (Extensible, Scalable Software for Evolving NetworkS) [76], that can be easily used to implement the algorithms on large networks. More details on ESSENS are presented in section 6.

We also used a parallel implementation of ESSENS for filtering biological networks [15]. Data intensive biological experiments are crucial in systems biology, for example in understanding cellular mechanisms and what happens in disease states. The output data from such experiments is huge in size, heterogeneous, and requires sophisticated computational algorithms for analysis. Correlation networks have been used previously for modeling and analysis of such massive data and using graph theory, structures were identified that correspond to key players in major cellular pathways. Filtering such networks i.e. reducing them in size while preserving the main structure of the network, has been shown to reduce noise and strengthen biological signals. The process of filtering biological networks is expensive computationally and thus we developed a parallel template to significantly reduce the computation time.

1.4 Main Contributions

The main contributions of this dissertation are:

- Application of group testing to synthetic and real-world networks to identify high centrality vertices.
- Application of noise models to networks in order to determine the stability of centrality rankings under network perturbation.
• Quantifying network stability under perturbations by examining local network properties such as the connectedness of top ranked vertices.

• Development of network analysis software (ESSENS) in order to implement the various algorithms used.

• Literature review of noise models that have been applied to networks and the methods used to quantify the stability, sensitivity, robustness, and reliability of networks.
2 Background

This section presents the relevant definitions and background on graph theory, centrality measures, group testing, the network noise model we use, parallel graph algorithms, existing network analysis software, and the input data we use.

2.1 Graph Theory

A network (also referred to as a graph) is defined by a pair of sets $G = (V, E)$, where $V$ is the set of vertices (entities) and $E$ is the set of edges (relationships). Each edge $e \in E$ is represented by a pair of vertices i.e. $e = (u, v)$ and $e$ corresponds to a relationship between $u$ and $v$. The vertices $u, v$ are known as the endpoints of $e = (u, v)$. A neighbor of $u$ is defined to be any vertex $v$ joined by an edge to $u$. A path, of length $l$, in a network $G$ is an alternating sequence $v_0, e_1, v_1, e_2, \ldots, e_l, v_l$ of vertices and edges, such that for $j = 1, \ldots, l$; $v_{j-1}$ and $v_j$ are the endpoints of edge $e_j$, with no edges or internal vertices repeated. A graph is undirected if $(u, v) \in E$ also implies that $(v, u) \in E$ and is unweighted if there are no values associated with the edges, that is all edges have the same importance.

2.2 Centrality Measures

The closeness centrality (CC) of vertex $v$ is defined as: $CC(v) = \frac{1}{\sum_{s \in V \setminus \{v\}} d(v, s)}$, where $d(v, s)$ is the distance between $v$ and $s$ (length of a shortest path between $v$ and $s$ in the network). Vertices that are overall closer to other vertices in the network will have higher CC values, thus CC measures importance with respect to accessibility i.e. from high CC vertices it is quicker (shorter) to reach the other vertices in the network.

The standard way of computing CC for one vertex is to perform a breadth-first-search in order to obtain the distances to all the other vertices, then take the reciprocal. For one vertex the complexity is thus $O(|V| + |E|)$, so for the whole network it is $O(|V|(|V| + |E|))$.

The standard algorithm is straightforward to parallelize; each processor is assigned a set of vertices and it calculates the closeness centrality value (sequentially) for each vertex it was assigned. There exist similar algorithms for computing closeness centrality on GPUs [66]. In addition, many parallel network analysis libraries such as STINGER [24], KDT [46], Parallel Boost Graph Library [32] also contain algorithms for computing closeness centrality (and betweenness). Although efficient use of data structures can reduce the cost, despite its simplicity, finding closeness centrality is still an expensive operation. An alternative is to use approximate computations such as those proposed by Eppstein and Wang [26]. Recently, Sariyuce et. al. presented a fast algorithm for updating closeness centrality on dynamic networks [61]. One common theme in these algorithms is that they all focus on finding all the closeness centrality values, rather than identifying...
which vertices have the highest values.

The betweenness centrality (BC) of vertex $v$ is defined as [28]:

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where $\sigma_{st}$ is the total number of shortest paths in $G$ between nodes $s$ and $t$, and $\sigma_{st}(v)$ is the total number of shortest paths in $G$ between $s$ and $t$ that pass through $v$. The most commonly used algorithm for computing betweenness centrality is the Brandes algorithm [9], which cumulatively computes the BC of every vertex while traversing the entire network. The Brandes algorithm has complexity $O(|V| \cdot |E|)$ on unweighted networks.

Computing BC values is a well studied problem. Approaches for reducing the time to compute betweenness centrality include parallel implementations of the Brandes method [3, 25], adaptive sampling for approximating BC on individual vertices [2] and iterative methods based on pivots [12]. All these methods still focus on computing the exact or nearly exact BC value of the vertices. However, as discussed earlier, the values are important only in their relative sense and it is the identity of the highest ranked BC vertices that we need not their actual values, that is the focus of this study.

2.3 Group Testing

This section provides an overview of group testing. To begin, a simple example is demonstrated in order to make the idea of group testing clear.

2.3.1 Simple Group Testing Example.

Before presenting our application of group testing to graphs, we begin with a simple example on a population $S = \{s_1, s_2, \ldots, s_8\}$ of $n = 8$ units of which $d = 1$ is defective. We can think of the units as being blood samples of which one is infected with some disease. We will show how the infected blood sample can be identified in $N = 3$ tests (as opposed to the exhaustive 8 tests where each sample is tested individually). Consider the $3 \times 8$ matrix $X$ below:

$$X = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix}$$

This matrix is a group testing design and specifies the groups (samples of the population) to be tested in the following way: row 1 ($X_1 = [0, 1, 0, 1, 0, 1, 0, 1]$) specifies the first group (subset of $S = \{s_1, s_2, \ldots, s_8\}$) to be tested. The $i^{th}$ position in the vector $X_1$ indicates that the unit $s_i$ is either part of the group to be tested (1) or not part of the group (0). Therefore, the group to be tested in the first test is simply $T_1 = \{s_2, s_4, s_6, s_8\}$.
the second group is $T_2 = \{s_3, s_4, s_7, s_8\}$, and the third group is: $T_3 = \{s_5, s_6, s_7, s_8\}$. Each test has a result component associated with it which indicates the presence of an infected unit (1) or the test result is negative (0). Suppose unit $s_6$ is defective. Then the result vector is:

$$Z = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Notice that $Z$ exactly matches the column for unit $s_6$ in the matrix $X$ thus allowing us to identify $s_6$ as the infected sample using only $\log_2 N = \log_2 8 = 3$ tests (as opposed to testing each unit individually resulting in 8 tests). The process is illustrated in Figure 2.

The method used in the example only works when it is known there is exactly one defective unit. If it is the case that there is more than one defective unit, then the technique will not work (the result vector will not match the appropriate column in the matrix) and a more sophisticated group testing design is needed.

### 2.3.2 Previous Work on Group Testing on Graphs

Classically, group testing has been used in the search for defective units among a population without taking into account any relationships the units may have with each other. In our problem, the population consists of vertices in a network and we are interested in identifying $d$ sensitive (defective) vertices. These are the $d$ highest centrality vertices. Since the BC measure takes into account shortest paths in the network and CC takes account of distances in the network i.e. both take into account the edges (which correspond to relationships between vertices) in the network, the group testing procedure must also take this into account. That is, we need to have a suitable way of grouping the vertices into a single entity on which we perform the test (calculate the centrality of it).

Group testing has been applied to graphs before, though not widely and for different problems. The first application of group testing to graphs, as far as we know, was for non-adaptive fault diagnosis for all-
optical networks [33]. This work, from 2007, used combinatorial group testing on graphs (corresponding to optical networks) to identify failed fiber links, failed optical amplifiers, and failed transmitters/receivers. The modification to the classical combinatorial group testing model is that now groups selected for each test must correspond to “permissible probes” in the network i.e. sets of adjacent edges which are part of a walk in the network in which each edge is traversed at most once in each direction. Bounds on the number of required tests for different optical network topologies are derived in the same paper [33]. More recently, group testing was applied on graphs corresponding to wireless sensor networks to identify congested links [11]. The authors refer to their problem as graph constrained group testing (GCGT), where the constraint is that each group to be tested must correspond to a set of vertices (in vertex group testing) or edges (in edge group testing) which are part of a walk in the graph. Unlike the previous work [33] where combinatorial group testing was used, in [11] random walks are used to generate the groups to be tested i.e. it is a random design with constraints dictated by the structure of the network.

2.4 Network Noise Model

There are quite a few models that have been developed previously in order simulate noise in networks. In section 4 we present a literature review of the various models.

The noise model proposed in [1] is the one implemented in our studies. It consists of several different types of network perturbation. The first type, and the one which we implement in our work, is the uniform perturbation model that is based on Erdős-Rényi random graphs. For a given parameter $\epsilon$, $0 \leq \epsilon \leq |V|$, an edge that is present in the original network has a probability of $\frac{\epsilon}{|V|}$ of being removed, and an edge that is not part of the original network has a probability of $\frac{\epsilon}{|V|}$ of being added. The second type is degree assortative perturbation which is based on the Chung-Lu random graph model. Edges in this model are selected for addition/deletion with a probability that is proportional to the degrees of the endpoints. That is, if the endpoints have high degree then it is more likely that the edge between them would be deleted or added if there is no edge between them already. The model is biased toward edges that have high degree endpoints. The third type is the link prediction based model where the results of a missing link prediction algorithm are used to identify the edges to add or remove. These different types of models were used in [1] for the perturbation of a wide variety of real world networks taken from SNAP [43] to analyze the effect on k-core (a widely use measure of vertex importance and connectedness). The results show that the top core is pretty sensitive to all of the different types of noise models used and the level of overlap with the original network varies non-monotonically with the level of perturbation. In our work, we have used the uniform perturbation model to analyze how the ranking of the top-$k$ vertices (for each centrality measure) changes as the noise level is increased. We also
vary the value of $k$, which generally has not been explored in previous studies on network noise and ranking.

### 2.5 Parallel Graph Algorithms

The increasing amount of applications of graph theory to real-world problems has made graph algorithms an essential and important research area. Graphs present a great method of modeling systems of inter-related entities and many practical problems can be modeled and solved using graphs and graph algorithms. The size of the problem can become extremely large, especially when the networks are massive, outgrowing the memory and computational capabilities of single processors, and thus parallel computing has to be implemented to significantly reduce the computation time, and split the memory load across processors, in order for the problem to become feasible. However, methods in parallel computing that have been successfully developed for mainstream parallel scientific applications, such as solving partial differential equations, are not always appropriate graph problems. We briefly discuss some of the challenges of parallel graph algorithms identified in [47].

The computations that are executed by a graph algorithm are based on the edge and vertex structure of the graph, which is not directly written in the code. Since the structure of the graph is not known a priori, and the algorithm has to work on a variety of graphs, the partitioning of the graph to use across processors becomes a challenge. A good partitioning of one graph, may not be a good partitioning of another graph. Graph problems can be highly unstructured and irregular. For example, one area of the graph may have many edges and interconnected vertices, while another part of the graph may be very sparse with few edges. Partitioning once again becomes a challenge; an unbalanced computational load can lead to a bottleneck and poor scalability. Due to the irregularity and unstructured nature of many real-world graphs, computation and data access tends to have poor locality. That is, during computation, data may have to be accessed that is in a different part of memory, which will take more time. Poor locality leads to significant increases in runtime. Another aspect of many graph algorithms is that the majority of time is spent on traversing the graph rather than executing large computations on the graph data. This means data access outweighs computation and so runtime could be negatively affected by waiting for memory reads.

### 2.6 Existing Network Analysis Software

There have been many network analysis software packages developed in recent years. For example there are general purpose implementations such as Cytoscape [65], NodeXL [67], JUNG [58], Gephi [6] and igraph [14], as well as those focused on specific applications such as EPIsimS [53] (epidemics), Organizational Risk Analyzer [60] (geospatial analysis) and Network WorkBench [79] (scientific collaborations). Many of the
networks used by researchers are massive in size, thus to cope with memory and computation requirements network analysis software has been introduced for the parallel domain.

Parallel Boost Graph Library [32] is a distributed memory based graph algorithm package that contains many of the relevant network analysis methods. Knowledge Discovery Toolkit (KDT) [46] is another distributed memory based package. KDT implements the algorithms as linear algebra functions. Shared memory based packages include SNAP [43] and NetworKit [68]. A vertex based approach to large graph analysis is provided by Pregel [52]. GraphLab [45] is a distributed graph processing implementation that is based on MapReduce.

For analyzing dynamic networks there are fewer implementations. Some of these are: GraphStreams [17] (a sequential package which allows quick network updates and dynamically keeps track of graph connectivity), GraphCT [23] (parallel implementation for dynamic networks), STINGER [24] (another parallel implementation for dynamic networks).

Other example of distributed graph processing frameworks include Pregel [52], and Spark GraphX [84]. Pregel [52] is a bulk synchronous message passing abstraction that iteratively runs the program at each vertex simultaneously. In each iteration, it gathers all messages from the previous iteration, and prepare messages for the next iteration. The program terminates when there exists no more messages and every vertex votes to halt. GraphX [84] is a large-scale graph processing framework developed on the top of Apache Spark. Since Spark is a data-parallel computation system, GraphX implements graph operations based on data-parallel operations available in Spark. The Gather-Apply-Scatter (GAS) model is a widely used paradigm for graph parallel processing. GraphX implements this paradigm on top of Spark data-parallel computation primitives.

2.7 Input Data

We have performed experiments on the following set of real-world networks collected from the DIMACS Implementation Challenge Set [4] and the Stanford Network Analysis Project [43]. The real-world networks we used are given in Table 1. We also performed experiments on randomly generated networks using the RMAT and LFR generators and the details are provided.

2.7.1 Real-World Networks

The real-world networks in Table 1 come from a wide variety of sources and their descriptions are given in the table. They vary widely in size ranging from 30 to 60,000 vertices and 70 to over 2 million edges.
Table 1: Test Suite of Real-World Networks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Vertices</th>
<th>Edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>34</td>
<td>78</td>
<td>A social network of friendships between 34 members of a karate club</td>
</tr>
<tr>
<td>Chesapeake</td>
<td>39</td>
<td>170</td>
<td>The ecosystem network in the Chesapeake Bay</td>
</tr>
<tr>
<td>AS1</td>
<td>3570</td>
<td>7750</td>
<td>Communication network of an autonomous system comprised of internet routers</td>
</tr>
<tr>
<td>AS2</td>
<td>6474</td>
<td>13895</td>
<td>Communication network of an autonomous system comprised of internet routers</td>
</tr>
<tr>
<td>Caida</td>
<td>16301</td>
<td>32955</td>
<td>Autonomous systems network</td>
</tr>
<tr>
<td>C. elegans</td>
<td>435</td>
<td>2025</td>
<td>the metabolic network of C. elegans species</td>
</tr>
<tr>
<td>Les Mis</td>
<td>77</td>
<td>254</td>
<td>Co-appearance network of characters in the same chapter of the novel Les Misers</td>
</tr>
<tr>
<td>GrQc</td>
<td>5242</td>
<td>14496</td>
<td>Collaboration network of authors submitting to Arxiv in the general relativity and quantum cosmology category</td>
</tr>
<tr>
<td>HepTh</td>
<td>9877</td>
<td>25998</td>
<td>Collaboration network of authors submitting to Arxiv in the high energy physics theory category</td>
</tr>
<tr>
<td>Power Grid</td>
<td>4941</td>
<td>6594</td>
<td>Network representing the topology of the Western States Power Grid of the United States</td>
</tr>
<tr>
<td>Brightkite</td>
<td>58228</td>
<td>214078</td>
<td>Social network</td>
</tr>
<tr>
<td>CondMat</td>
<td>23133</td>
<td>93478</td>
<td>Condensed matter collaboration network</td>
</tr>
<tr>
<td>Gnutella</td>
<td>62986</td>
<td>147892</td>
<td>Peer-to-peer file sharing network</td>
</tr>
<tr>
<td>Railway</td>
<td>301</td>
<td>1224</td>
<td>Network corresponding to the Indian railway system</td>
</tr>
<tr>
<td>Football</td>
<td>115</td>
<td>613</td>
<td>Network of American football games between Division IA colleges</td>
</tr>
<tr>
<td>Email</td>
<td>1133</td>
<td>5451</td>
<td>Network of e-mail interchanges between members of the University Rovira i Virgili</td>
</tr>
<tr>
<td>Dolphins</td>
<td>62</td>
<td>159</td>
<td>Social network of frequent associations between a community of 62 dolphins</td>
</tr>
</tbody>
</table>

2.7.2 Random Networks - RMAT

We have also used synthetic networks obtained using the RMAT generator, which is a widely used graph generation software that provides a simple and quick method for generating large graphs [10]. The generator only requires a few parameters for generation, it has an elegant design that is straightforward to use, and it can be easily implemented in parallel in order to generate larger graphs faster. The main goal for the development of the RMAT generator was to produce synthetic networks that represent the structure of certain real-world networks. RMAT networks can be generated to have power-law degree distributions, parameters can also be set so that degree distribution of the resulting network matches the distribution of another existing network. Networks obtained using the RMAT generator have been used in several research areas such as social network analysis, network monitoring, biology, and as benchmarks for graph algorithms. The RMAT networks which we used range in size from 1000 to 200,000 vertices and from 170,000 to 4,000,000 edges.

2.7.3 Random Networks - LFR

Another network generator known as LFR [41] was also used to obtain synthetic networks for the experiments. With LFR it is possible to generate networks which exhibit a strong community structure, meaning the networks contains different groups of vertices (communities) which are highly connected between themselves and have almost no connections to vertices in other communities. Many real-world networks, such as Facebook, have this community structure property and community detection networks is a widely researched topic [55]. We used LFR to generate networks that are scale-free and with strong communities (using parameter $\mu = .1$ and other parameters as given in default settings). The size of the networks we generated ranges from 1000 to 10000 vertices.
3 Group Testing on Networks

In this section we present a more detailed description of our group testing method, superimposed codes, our results in group testing on networks for high centrality vertices, and the issues involved with group testing on networks. Some of this work has been published by us previously in [74], [73], and [75].

3.1 Method

In our problem, we apply group testing to graphs in a way that is different from previous work. The designs we use are directly from classical combinatorial group testing and there are no constraints (as there are in [33] and [11]) placed on the actual group to be tested. However, for each given test, the group of selected vertices are combined into one "supervertex" whose set of neighbors (other vertices it is connected to via one edge) is the union of the sets of neighbors of its constituent vertices. That is, given a graph $G = (V, E)$, suppose that in test $i$ we have the set of vertices to be grouped and tested: $T_i = \{v_1, v_2, ..., v_g\} \subset V$. Let $v_T$ denote the supervertex made by combining all vertices in $T_i$ and let $A_{v_j}, j = 1, 2, ..., g$ denote the set of neighboring vertices for vertex $v_j$ i.e. $A_{v_j} = \{u \in V : (v_j, u) \in E\}$. Then the set of neighboring vertices of the supervertex corresponding to $T_i$ is simply $A_{T_i} = \{u \in V : \exists v_j \in T_i, (v_j, u) \in E\} = \bigcup_{j=1}^{g} A_{v_j}$. When performing this grouping of vertices we are essentially compressing the original graph $G$ into a smaller (has $g - 1$ less vertices and less edges) graph $G_{T_i}$ on which we perform the centrality calculations. In fact, for test $T_i$ we only need to calculate the centrality of the supervertex $v_T$ in graph $G_{T_i}$ and this value will determine whether the result is positive (1) or negative (0).

The reasoning in our method is that when high centrality vertices are part of a group, and the group is combined into a supervertex as described previously, then the centrality of the supervertex should be high also. Similarly, if the group contains vertices which are not in the set of $d$ highest centrality vertices, then the centrality of the supervertex for that group should also be relatively low i.e. below some threshold $\tau$. When the centrality of the supervertex in some test $T_i$ is above threshold $\tau$ then the result of the test will be positive (1), and when the result is not more than $\tau$, the result of the test will be negative (0). One important aspect in our method is the selection of an appropriate threshold so that the $d$ highest centrality vertices can be determined. Also, our method singles out the highest $d$ centrality vertices, so ideally the network should have centrality distributions where there are not more than $d$ vertices with outstandingly high centrality (so they can be singled out more easily) and the rest of the vertices should all have relatively low centrality values.
3.1.1 Theoretical Background

There are two main types of group testing algorithms; adaptive and nonadaptive, though there are algorithms that can be thought of as a combination of both, such as two-stage group testing. In adaptive group testing, the result of a test gives some information about what units to group in the next test. A well known example is a binary search group testing algorithm where in the first test half of the population is tested, then from the result it is determined which quarter to test next, then which eighth and so forth until the one defective unit is left. All tests must be executed sequentially in the correct order. In nonadaptive group testing, all of the tests to be done are specified beforehand without knowing the outcomes of other tests. The group to be tested does not depend on the result of the previous test. Nonadaptive group testing is employed when there is a constraint on time (since all tests can be carried out simultaneously, thus reducing the time) or when there is a cost constraint (obtaining information from other tests during the experiment will cost more money than waiting until all testing is complete) [16]. Nonadaptive group testing algorithms can be represented by a binary matrix, where the columns of the matrix correspond to each unit in the population and the rows correspond to a test and determine which units of the population are to be tested. The binary matrix has constraints imposed on the component-wise Boolean sum of any $d$ (or up to $d$) columns. Such matrices fall under Superimposed Coding Theory and are known as superimposed codes [38].

A superimposed code of length $N$ and size $n$ is represented as a binary $N \times n$ matrix, $X$. Let $x_{i,j} \in \{0, 1\}$ denote the element in row $i$ and column $j$ of $X$ and let $x_j$ denote the $j^{th}$ column of $X$. The Boolean-OR sum of any $k$ columns $x_{j_1}, x_{j_2}, \ldots, x_{j_k}$ is:

$$f(x_{j_1}, x_{j_2}, \ldots, x_{j_k}) = \begin{bmatrix} x_{1,j_1} \lor x_{1,j_2} \lor \ldots \lor x_{1,j_k} \\ x_{2,j_1} \lor x_{2,j_2} \lor \ldots \lor x_{2,j_k} \\ \vdots \\ x_{N,j_1} \lor x_{N,j_2} \lor \ldots \lor x_{N,j_k} \end{bmatrix}$$

where $\lor$ is the Boolean-OR operation i.e. $0 \lor 0 = 0, 0 \lor 1 = 1, 1 \lor 0 = 1, 1 \lor 1 = 1$.

A column $x_j$ covers column $x_i$ if $f(x_j, x_i) = x_j$. Code $X$ has strength $d$ if and only if the Boolean-OR sum of any $d$ columns does not cover any other column [22]. The weight, $w(x_j)$, of column $x_j$ is the number of ones in the column. The minimum weight $w = \min_{1 \leq j \leq n} w(x_j)$. The intersection, $\lambda(x_j, x_i)$, between two columns $x_j, x_i$ is the number of positions in which both $x_j$ and $x_i$ have a 1. The maximum intersection $\lambda = \max_{1 \leq j \neq i \leq n} \lambda(x_j, x_i)$. The Kautz-Singleton Bound [38] states that $d \geq \left\lfloor \frac{w - 1}{\lambda} \right\rfloor$. This results shows that the minimum weight $w$ and maximum intersection $\lambda$ are sufficient to obtain a lower bound on the strength $d$ of the code. D’yachkov and
Rykov (22) proved that as $n \to \infty$ and $d \to \infty$ with $d \leq \log_2 n$, the minimum number of tests $N$ is bounded by: $\Omega\left(\frac{d^2}{\log_2 d} \log_2 n\right) \leq N \leq O\left(d^2 \log_2 \frac{n}{d}\right)$.

### 3.1.2 Latin Square Superimposed Code Construction

In our experiments, we used a superimposed code constructed from a Latin square, which is guaranteed to find at least 2 defective units [38]. The superimposed code $X$ is created as follows: given a population of $n$ units, create a finite set of contiguous integers $\{1, 2, \ldots, l\}$, where $l = \lceil \sqrt{n} \rceil$. Then create an $l \times l$ Latin square matrix $L$, where $L_{ij} \in \{1, 2, \ldots, l\}$ such that each element from $\{1, 2, \ldots, l\}$ appears exactly once in any given row and column (left diagram in Figure 3(a)).

We construct a group testing design matrix $X$ from a Latin square $L$ in the following way; the first 2 positions in any given column in $X$ are coordinates (row and column) in $L$ and the 3rd position is the element in $L$ at those coordinates (middle diagram in Figure 3(a)). We then encode each integer in $X$ in a binary format. Each integer is coded as a binary vector of length $l$, where for integer $i$, the vector has zero in all positions, except at position $i$ which has one. In other words, the binary representation of integer $i$ is the $i^{th}$ row (or column) of an $l \times l$ identity matrix (right hand diagram in Figure 3(a)). Once the elements of the coding matrix are transformed to their binary form, the total number of tests is equal to the number of rows in the $X$ matrix, which is $3 \lceil \sqrt{n} \rceil$, for a population of $n$ units (Figure 3(c)).

Due to the Latin square construction, any two columns in $X$ can intersect in at most one position. Defective units are those whose value is more than the user selected threshold. We find the minimum weight $w = 2$ and maximum intersection $\lambda = 1$ then use the Kautz-Singleton Bound to get a guaranteed value for the strength parameter $d \geq \lfloor \frac{w-1}{\lambda} \rfloor = \lfloor \frac{1}{2} \rfloor = 2$. Although ideally in non-interactive cases the final results of the Latin-Squares method should not vary, the composition of the groups can change according to how the vertices are ordered.

An example of how Latin square group testing can be used on a small network is given in Figure 3. The network consists of 16 vertices, composed of two cliques of 8 vertices each. The Latin Square is therefore created for a 4 by 4 matrix by right shifting the numbers $\{1, 2, 3, 4\}$ in each row. The superimposed code has 16 columns and 12(3*4 rows), where the first row gives the row in the Latin Square, the second row gives the column and the third row gives the value. This matrix is then expanded to its binary form. Each row denotes a test, presence of a 1 indicates that a vertex will be included in the test. For example, test 2 contains only the vertices $\{4, 5, 6, 7\}$, which are combined as a supervertex. The threshold is set to 65 and all tests where the BC value of the supervertex is higher than 65 are marked as positive (colored red). The result vector has 1 for positive tests and 0 for negative ones. Note that the Boolean-OR of the columns 4 and 8 is exactly the same.
Figure 3: **Example of Group Testing Using Latin Squares.** (a): Construction of coding matrix using a 4 by 4 Latin square. (b): A sample graph of two-8-cliques connected by one edge. The final matrix given in (c), allows at most 16 units to be tested. The threshold is set to 65. Clearly vertices 4 and 8 are the ones with highest BC, as given by the Results column in (c).

as the resultant vector. Therefore vertices 4 and 8 are the high BC vertices.

Similarly, we can use the same design to identify high closeness centrality vertices as illustrated in Figure 4. The network consists of 16 vertices, composed of two spokes of 8 vertices each, and the centers of the spokes are connected with each other. Clearly vertices 0 and 8 have the highest centrality. Their closeness value is .045, while that of all the other vertices is .027. The Latin Square is created for a 4 by 4 matrix by right shifting the numbers \{1,2,3,4\} in each row. The coding matrix has 16 columns and 12(3*4 rows), where the first row gives the row in the Latin Square, the second row gives the column and the third row gives the value in the corresponding row and column. This matrix is then expanded to its binary form. Each row denotes a test, presence of a 1 indicates that a vertex will be included in the test. For example, test 2 contains only the vertices \{4,5,6,7\}, which are combined as a supervertex. The threshold is set to .0523 and all tests where the CC value of the supervertex is higher than .0523 are marked as positive (colored red). The result vector has 1 for positive tests and 0 for negative ones. Note that the Boolean-OR of the columns 0 and 8 is exactly the same as the resultant vector. Therefore vertices 0 and 8 are the high ranked vertices.

The group testing design we use is non-adaptive, meaning the tests are independent and thus can be executed simultaneously. To exploit this, we parallelize our group testing algorithms, splitting tests across
processors in order to further reduce runtime. An example of this process is illustrated in Figure 5.

When identifying high centrality vertices in networks, the standard brute-force method is to calculate the centrality value of every vertex, then sort the list to find the highest ranked vertices. Indeed, most of the centrality algorithms are used to calculate the centrality values for all of the vertices. For betweenness centrality, the most common algorithm [9] has to calculate the values for all of the vertices and cannot find the value of some specified vertex without having to compute the rest. In our problem of identifying just the highest ranked vertices, a key point to note is that we are interested in the identity of the highly ranked vertices, not their actual centrality values. Using group testing, we can find the highest ranked vertices by calculating centrality values for carefully selected groups of vertices in order to determine the presence/absence of highly ranked vertices within the group. The main motivation of using this method is to significantly reduce the number of centrality calculations needed to identify the highest ranked vertices. Applying group testing methods on networks to find vertices of high centrality can be broken down into three major stages. These are:

Stage 1. Creating Appropriate Groups of Vertices. Since we are using Latin square based group testing the number and composition of the groups can be obtained from the number of vertices in the network. The coding matrix will contain size $3\lceil \sqrt{|V|} \rceil$ groups to be tested and each group will contain $\lceil \sqrt{|V|} \rceil$ vertices. We however do not create and store this matrix explicitly as that would be very memory intensive. Instead, based
on the number of vertices and the group number we can identify the vertices that will be part of the group. This information is used in the second step to compute the centrality per group of vertices. The time required for this step is proportional to the number of groups, and is \(O(3\lceil \sqrt{|V|} \rceil)\).

**Stage 2. Computing Centrality of the Groups.** Ideally the vertices in each group are combined together to form a supervertex and then the centrality is computed for the supervertex in each group. While computing closeness centrality, we do not explicitly combine the vertices to form the super vertex. Instead, we execute the BFS such that rather than one source, the root consists of multiple sources (the vertices in the group being the sources). Thus the execution time is \(O(|V| + |E|)\) per group and the total time for this step is \(O(3\lceil \sqrt{|V|} \rceil(|V| + |E|))\). For computing betweenness centrality we can either use the Brandes algorithm on the smaller network (since vertices have been grouped) or we can implement the single vertex BC algorithm we have developed.

**Stage 3. Identifying High Ranked Vertices.** Each group produces a centrality value for its corresponding supervertex. Once the centrality values for all the groups are obtained, we select a threshold such that the groups with centrality values above or equal to the threshold are marked as one and the rest are marked as zero. This process creates a resultant binary vector. We then identify which two columns in the coding

Figure 5: **Illustration of Parallel Process.** The figure shows the parallel implementation of our group testing algorithm. Each group is sent to a different processor which calculates the value of the test.
matrix are covered by the resultant vector. The vertices corresponding to those columns are the high centrality vertices.

3.2 Betweenness Centrality Results

We executed our group testing algorithm for BC on the networks shown in Table 2. In order to evaluate the accuracy, we compare how many of the nodes identified to be high ranking using group testing also have high rank when the exact BC values are computed. We deem the group testing method successful if group testing is successful in correctly identifying the top 2 vertices. The results are given in Table 2. Out of the ten networks, group testing was successful in six networks (top six rows of the table), and found low ranked (below rank 10) vertices for the other four (the last four rows of the table).

An interesting case is the Les Mis dataset where group testing was successful in finding the top vertex and then identified the vertex with rank 10 as the next highest. Since this is the network of interactions between characters in the novel Les Miserables, we could further investigate why group testing failed. The top vertex corresponds to Jean Valjean, the protagonist in the novel. The second highest BC vertex as computed by the Brandes method corresponds to Bishop Myriel, who only appears in 27 chapters out of a total 365. The Bishop is one of the few characters in the early chapters of the book who interacts with the protagonist and therefore is a vertex that links the characters in the first chapters with the rest of the novel. This character is almost an articulation point in the network, and therefore has high BC value.

Our group testing method, however, failed to find this second highest BC vertex and got the tenth highest vertex instead. This vertex corresponds to Tholomyes, a character who appears in only 9 chapters in the entire novel. This vertex is also an almost articulation point, and is connected with the network not by the protagonist but by a supporting character Fantine (BC rank 6).

In the group testing, if the Bishop vertex was often combined with other vertices that were also connected to the protagonist, then these other vertices duplicated the contribution of the Bishop vertex to the high BC value. On the other hand, the tenth highest vertex since it was not connected via the protagonist could contribute more to the BC value because it showed more unique connections. This is an example where combining several medium ranked BC vertices that cover different regions of the networks can lead to a higher values than the actual high BC vertex covering the same region.

3.2.1 Scalability Results

We also present the scalability results for the parallel group testing for BC implementation. The experiments were performed on the Firefly cluster at the Holland computing center. The cluster consists of 280 nodes
Table 2: Finding High BC Vertices Using Group Testing on Real-World Networks. The best threshold and the vertices obtained using that threshold are given. The vertices are represented by their rank, as per their BC values obtained using the Brandes method.

<table>
<thead>
<tr>
<th>Name</th>
<th>Vertices</th>
<th>Edges</th>
<th># of Tests</th>
<th>Threshold</th>
<th>High BC Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>34</td>
<td>156</td>
<td>18</td>
<td>55%</td>
<td>1st, 2nd</td>
</tr>
<tr>
<td>Chesapeake</td>
<td>39</td>
<td>340</td>
<td>21</td>
<td>30%</td>
<td>1st, 2nd</td>
</tr>
<tr>
<td>AS20000102</td>
<td>6474</td>
<td>13233</td>
<td>243</td>
<td>12%</td>
<td>1st, 2nd, 3rd</td>
</tr>
<tr>
<td>AS20000101</td>
<td>3570</td>
<td>7391</td>
<td>180</td>
<td>16%</td>
<td>1st, 2nd</td>
</tr>
<tr>
<td>Caida</td>
<td>16301</td>
<td>65910</td>
<td>384</td>
<td>21%</td>
<td>1st, 2nd, 3rd</td>
</tr>
<tr>
<td>C. Elegans</td>
<td>453</td>
<td>4050</td>
<td>66</td>
<td>35%</td>
<td>1st, 2nd, 4th, 10th + 3 low ranked</td>
</tr>
<tr>
<td>Les Mis.</td>
<td>77</td>
<td>508</td>
<td>27</td>
<td>45%</td>
<td>1st, 10th, +3 low ranked</td>
</tr>
<tr>
<td>GrQc</td>
<td>5242</td>
<td>28980</td>
<td>219</td>
<td>80.3%</td>
<td>20 low ranked</td>
</tr>
<tr>
<td>HepTh</td>
<td>9877</td>
<td>51971</td>
<td>300</td>
<td>76%</td>
<td>6 low ranked</td>
</tr>
<tr>
<td>Power Grid</td>
<td>4941</td>
<td>13188</td>
<td>213</td>
<td>84%</td>
<td>6 low ranked</td>
</tr>
</tbody>
</table>

running two AMD Quad core processors and 871 nodes running two dual core Opteron processors.

We executed the group testing in parallel using a master-worker model, where each worker executes one test at a time and communicates the results to the master. MPI was used to exchange the jobs and information between the master and the worker processors. Given in Figure 6 are the strong scalability results of four of the larger networks from our test suite. The results show that the parallel implementation is very scalable. Furthermore the parallel algorithm is designed such that we obtain the same results regardless of the number of processors used.

3.2.2 Algorithm for Betweenness Centrality of a Single Vertex

In order to efficiently apply our group testing algorithm, we have been developing an algorithm that computes the BC of one vertex only. The efficiency of the algorithm relates to the largest size and the number of the strict cycles in the graph. If the graph is chordal, i.e. the largest unbroken cycle is of length three, then we can compute the BC of a designated vertex in time proportional to execute a breadth first search. For larger cycles, in the worst case, we have to execute a BFS for each cycle. Therefore, if the number of strict cycles in the graph greater than 3 is \(q\), then computing the BC of a vertex would take time \(O(q \times V)\). Therefore we can compute the BC of a single vertex in shorter time than computing the values for all the vertices.

We now briefly describe the algorithm. Given a network \(G = (V, E)\) we wish to find the BC only for the vertex \(v\). We begin by performing an initial BFS having \(v\) as the root vertex. The vertices at level 1 i.e. the direct neighbors of \(v\) are designated as branch roots and we keep track of which branch each subsequent vertex belongs to. We also keep track of the level in the BFS tree for each vertex and the vertex multiplicity. Any
time a vertex is repeated at the same level in the BFS tree and in different branches, we update its multiplicity
and queue it for a subsequent BFS which has the repeated vertex as the root. Repeats in the same branch or
at different levels are not taken into account.

Once the first BFS is complete we perform a path count. This is done by first checking pairs of different
branches, if any 2 branch roots have an edge between them then one of the roots are enqueued for a subsequent
BFS and we will not consider any paths originating at one branch and ending in the other (since it is shorter
not to go through the root node \(v\) for which we are calculating BC). For pairs of vertices across different
disconnected branches we check first if the 2 vertices are directly connected by an edge. If so we enqueue
one of them for a subsequent BFS and disregard the pair since there is no shorter or equal length path between
them going through the root \(v\). For all pairs across different disconnected branches that are themselves not
connected we store the information as a possible path between the two vertices that goes through \(v\). The
distance is given by the sum of their levels and the number of paths passing through \(v\) between the two
vertices is obtained by multiplying their individual multiplicities.

Now we perform a subsequent BFS for each vertex enqueued in the new source queue. This is done
exactly the same as the first BFS except we do not queue any repeated vertices, though we still keep track of
multiplicities, and we do not take into account (do not add to BFS tree) any vertices that have been previously
used as a BFS root (including \(v\)). Therefore the subsequent BFS trees get progressively smaller. Upon
completion of each subsequent BFS we cycle through the paths currently stored. If a path has source and
destination vertices that are in different disconnected branches in the BFS just completed we compare the
original distance to the new distance in the new BFS tree. If the original is larger we mark out the path since

Figure 6: **Strong Scalability Results of Group Testing.**
it is not shorter to go through \( v \). If original is the same as the new one then we update the number of paths not passing through \( v \) between the pair of vertices (by multiplying their multiplicities).

Finally the BC of \( v \) is computed by calculating the following fraction: \( \frac{\text{Passing}}{\text{Passing} + \text{NotPassing}} \) for each remaining path and summing over all remaining paths to get the total BC.

We are still in the process of optimizing the algorithm in order to use it in the group testing method to identify high BC vertices.

### 3.3 Closeness Centrality Results

Our experiments for closeness centrality were performed on a set of 4 real world networks (obtained from the Stanford Network Analysis Project [43]) and 4 synthetic networks obtained using the R-MAT generator [10]. The input parameters for R-MAT include the size of graph in terms of the number of vertices and edges, and a set of four probabilities that sum to one. The probabilities determine the structure of the network. Table 3 provides an overview of the properties of our test suite.

Table 4 provides the number of tests that were done, the threshold and whether the identified vertices were indeed high rank. Although Latin square guarantees the top-2 high ranked elements, in cases where the closeness centrality of the second highest vertex was not as significantly high from the other vertices, we obtained only the first. Conversely, sometimes due to the structure of the network we obtained the 1st, 2nd and the 3rd highest vertices. The only major failure in this set was that of the Gnutella network. We only found low centrality vertices in this cases. We posit that as a file-sharing network, the vertices have similar centrality values. By examining the distribution of centrality values for Gnutella, indeed it is the case that the top values are similar and decrease gradually.

#### 3.3.1 Parallel Group Testing for Closeness Centrality Algorithm

A simple yet effective and scalable parallel algorithm to compute the closeness centrality of all the vertices in an undirected network is to compute the closeness centrality of each vertex in parallel. This involves computing a breadth first search (BFS) with each vertex as a root. The complexity of a BFS algorithm is \( O(|V| + |E|) \). Therefore with \( p \) processors, the time taken to compute the closeness centrality of all the vertices would be \( O\left(\frac{|V|}{p}\right)(|V| + |E|) \). The pseudocode of the algorithm is shown below.
Algorithm 1 Parallel Closeness Centrality. **Input:** A connected graph $G = (V, E)$. **Output:** Closeness Centrality of Each Vertex

1: for $u \in V$ do in parallel
2: \hspace{1em} $CC[u] \leftarrow 0$
3: \hspace{1em} $Q \leftarrow Q.push\_back(u)$
4: for $v \in V$ do
5: \hspace{2em} $dist[v] \leftarrow 0$
6: \hspace{2em} $Visited[v] \leftarrow FALSE$
7: \hspace{1em} $Visited[u] \leftarrow TRUE$
8: while $Q$ is not empty do
9: \hspace{2em} $s \leftarrow Q.pop()$
10: \hspace{3em} for all $n$ that are neighbors of $s$ do
11: \hspace{4em} if $Visited[n] = FALSE$ then
12: \hspace{5em} $Visited[n] \leftarrow TRUE$
13: \hspace{4em} $Q \leftarrow Q.push\_back(n)$
14: \hspace{3em} $dist[n] \leftarrow dist[s] + 1$
15: \hspace{2em} $CC[u] \leftarrow CC[u] + dist[n]$
16: $CC[u] \leftarrow 1/CC[u]$

Table 3: Test Suite of Networks for Closeness Centrality.

<table>
<thead>
<tr>
<th>Name</th>
<th>Vertices</th>
<th>Edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caida</td>
<td>16301</td>
<td>65910</td>
<td>Autonomous systems network</td>
</tr>
<tr>
<td>Brightkite</td>
<td>58228</td>
<td>428156</td>
<td>Social network</td>
</tr>
<tr>
<td>CondMat</td>
<td>23133</td>
<td>186956</td>
<td>Condensed matter collaboration network</td>
</tr>
<tr>
<td>Gnutella</td>
<td>62586</td>
<td>295784</td>
<td>Peer-to-peer file sharing network</td>
</tr>
<tr>
<td>RMAT-B1-17</td>
<td>106714</td>
<td>2035180</td>
<td>Parameters: 17, 17, .55, .15, .15, .15</td>
</tr>
<tr>
<td>RMAT-B4-17</td>
<td>83687</td>
<td>1686088</td>
<td>Parameters: 17, 17, .65, .1, .1, .15</td>
</tr>
<tr>
<td>RMAT-B1-18</td>
<td>209026</td>
<td>4098078</td>
<td>Parameters: 18, 18, .55, .15, .15, .15</td>
</tr>
<tr>
<td>RMAT-B4-18</td>
<td>161532</td>
<td>3448662</td>
<td>Parameters: 18, 18, .65, .1, .1, .15</td>
</tr>
</tbody>
</table>

Table 4: Finding High CC Vertices Using Group Testing on Real-World and Synthetic Networks. The best threshold and the vertices obtained using that threshold are given. The vertices are represented by their rank.

<table>
<thead>
<tr>
<th>Name</th>
<th>Vertices</th>
<th>Edges</th>
<th># of Tests</th>
<th>Threshold</th>
<th>High CC Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caida</td>
<td>16301</td>
<td>65910</td>
<td>384</td>
<td>84%</td>
<td>1st, 2nd, 3rd</td>
</tr>
<tr>
<td>Brightkite</td>
<td>58228</td>
<td>428156</td>
<td>726</td>
<td>83%</td>
<td>1st, 3rd</td>
</tr>
<tr>
<td>CondMat</td>
<td>23133</td>
<td>186956</td>
<td>459</td>
<td>96%</td>
<td>1st</td>
</tr>
<tr>
<td>RMAT-B1-17</td>
<td>106714</td>
<td>2035180</td>
<td>981</td>
<td>85%</td>
<td>1st</td>
</tr>
<tr>
<td>RMAT-B4-17</td>
<td>83687</td>
<td>1686088</td>
<td>870</td>
<td>85.7%</td>
<td>1st, 7th</td>
</tr>
<tr>
<td>RMAT-B1-18</td>
<td>209026</td>
<td>4098078</td>
<td>1374</td>
<td>87%</td>
<td>1st</td>
</tr>
<tr>
<td>RMAT-B4-18</td>
<td>161532</td>
<td>3448662</td>
<td>1206</td>
<td>85%</td>
<td>1st</td>
</tr>
<tr>
<td>Gnutella</td>
<td>62586</td>
<td>295784</td>
<td>753</td>
<td>-</td>
<td>All low rank at any threshold</td>
</tr>
</tbody>
</table>
### Table 5: Execution Time (sec) of The Networks.

<table>
<thead>
<tr>
<th>Name</th>
<th>2 Cores Normal</th>
<th>4 Cores GT</th>
<th>8 Cores Normal</th>
<th>16 Cores GT</th>
<th>32 Cores Normal</th>
<th>64 Cores GT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caida</td>
<td>3.25</td>
<td>0.23</td>
<td>1.53</td>
<td>0.48</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CostMat</td>
<td>132.28</td>
<td>22.74</td>
<td>14.35</td>
<td>14.75</td>
<td>10.96</td>
<td>10.96</td>
</tr>
<tr>
<td>Brightkite</td>
<td>90.55</td>
<td>35.50</td>
<td>42.59</td>
<td>32.54</td>
<td>3.54</td>
<td>3.54</td>
</tr>
<tr>
<td>CondMat</td>
<td>132.25</td>
<td>9.22</td>
<td>7.34</td>
<td>5.03</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Brightkite</td>
<td>90.55</td>
<td>35.50</td>
<td>42.59</td>
<td>32.54</td>
<td>3.54</td>
<td>3.54</td>
</tr>
<tr>
<td>Brightkite</td>
<td>90.55</td>
<td>35.50</td>
<td>42.59</td>
<td>32.54</td>
<td>3.54</td>
<td>3.54</td>
</tr>
<tr>
<td>Brightkite</td>
<td>90.55</td>
<td>35.50</td>
<td>42.59</td>
<td>32.54</td>
<td>3.54</td>
<td>3.54</td>
</tr>
<tr>
<td>Brightkite</td>
<td>90.55</td>
<td>35.50</td>
<td>42.59</td>
<td>32.54</td>
<td>3.54</td>
<td>3.54</td>
</tr>
<tr>
<td>Brightkite</td>
<td>90.55</td>
<td>35.50</td>
<td>42.59</td>
<td>32.54</td>
<td>3.54</td>
<td>3.54</td>
</tr>
</tbody>
</table>

#### 3.3.2 Performance

We now present results on the execution time of the algorithms. We ran the experiments on the Tusker supercomputer at the Holland Computing Center at UNO which consists of 106 AMD Interlagos-based nodes (6784 cores) interconnected with Mellanox QDR Infiniband with 256 GB per node. In this set of experiments we used distributed memory (MPI) as the programming paradigm. However, since each processor reads the network and the computations are nearly independent a shared-memory model would have been equally, if not more, effective. The tests were performed on 2, 4, 8, 16, 32 and 64 cores. Figure 7 provides the ratio of the time for executing the standard parallel closeness centrality algorithm to the time taken to execute our group testing algorithm. The actual times are given in Table 5. We also provided a chart of the timings on 2 -cores Figure 8, to provide a visual estimate of the spectacular improvement due to using group testing.

From the results we see that the group testing method is significantly faster than the parallel closeness centrality computation. The best improvement is over 50 times and even the smallest improvement is more than 7 times. We observe that the improvement is even more for larger networks. This is to be expected because as discussed earlier, Latin square group testing improves about $O(\sqrt{|V|})$ times over the parallel implementation. We also observe that the improvement decreases as the number of cores increase. Again this is to be expected. Group testing requires two communication phases, as opposed to one (or none) of the parallel closeness centrality computation. Therefore using more cores increases the communication overhead in group testing. In Figure 9 we provide strong scalability results of the group testing algorithm over the networks. The results show that the algorithm is extremely scalable as well as very fast. In fact computing group testing for the Caida network took less than a second on 64 cores.

#### 3.4 Issues in Group Testing

Although group testing seems a natural match for finding important or sensitive vertices in a network, there exist several issues (some related to group testing itself, and some due to special properties of the network) that can affect the accuracy and performance of the method. Some of the issues and our solution to them are discussed below.
3.4.1 False Positive Results Due to Interaction Among Vertices

Group testing was originally designed for non-interacting samples i.e. samples that do not combine to change their characteristics. For example, two non-infected blood samples when combined, do not create an infected sample. But this is not the case for vertices in a network. Two (or more) vertices with low to medium centrality values can together combine in a sample to create a group with high centrality.
We have observed two types of false positives; in the first type, the vertices with very low centrality can be falsely identified if due to their placement in the superimposed code, they are always in the same group with at least one very high centrality vertex. These false positives can be eliminated by permuting the vertices in the graph to create a new set of groups. Alternatively, we can use known topological characteristics, for example vertices with clustering coefficient 1 have 0 betweenness centrality, to cull out some of these false positives.

The second type is when groups of vertices with medium centrality values combine together to give positive results. There is a much lower chance of this happening because the positive results also have to cover the appropriate columns of the vertices. Therefore if both high and medium BC vertices get covered then there would be an avalanche of positively marked vertices which would signal the presence of many false positives. In our experiments, we select thresholds to restrict the number of positive results to at most 7 vertices, though sometimes this is not possible, especially in the unsuccessful cases where many more positive results are identified, even at the higher thresholds.

There are also cases where only the vertices with medium centrality values show up as positive, their group centrality outranking the vertices with individually high centrality values. This happens when the original high centrality vertices are by themselves not high enough to counterbalance the cumulative effect of mid-ranked vertices. Our method cannot distinguish false positives in this case. However, we have also observed that in these problem instances, the rankings of the high centrality vertices change under small perturbations to the network. Thus, for these problems, the ranking is not very stable, and ranking the vertices by their centrality values would be more academic than utilitarian.
3.4.2 Selection of Threshold

The threshold value for a given group testing design determines which tests are classified as positive (above threshold and thus contain sensitive units) and which are negative (below threshold). Selection of correct threshold is a problem inherent to group testing itself, and even more important when applied to finding high centrality vertices, due to the probability of achieving false positive results.

In our experiments, we start by selecting the threshold to be 3rd highest centrality value obtained by group testing (so we have at least three 1s in the results column and therefore can cover a column in the superimposed code) and then keep on decreasing the threshold by setting it to be the 4th highest, then 5th, then 6th and so forth until we obtain enough positive results to cover the columns corresponding to a highly ranked vertices. This strategy is currently based on trial and error and we are working on a theoretical method to select the threshold a priori. The Latin Square method guarantees two defective units and we have seen that by slightly decreasing the threshold we can obtain 3-4 high centrality vertices. We therefore keep on decreasing the threshold until about 7 columns are covered. Some of these columns are false positives, but they can be eliminated using the techniques described above.

The value of the threshold does not affect the complexity of our algorithm, only the final results. Thus, even if we select a very low threshold, the runtime will be the same but the final result will have too many identified vertices (many false positives).

In the next sections we present our literature review on network noise models and our experiments with modeling noise in networks. From our experiments there is an evident connection between networks which are successful in our group testing experiments and those which are stable under network noise. Before presenting our findings on the link between group testing and stability under noise, we first present a review on the existing noise in networks research.
4 Network Noise Models - Literature Review

In this section, we present a review of the existing research of noise in networks. In general noise is defined as changes in the structure of the network, by adding/deleting vertices/edges. Although we found several papers tackling this problem, there was no standard definitions for the effect of noise in networks, what constitutes a noise model or how to measure the changes in the network. As part of this dissertation we have classified the effect of noise in networks into the four categories as follows:

1. **Stability** measures whether the *ranking* of the properties, such as the centrality metrics, change with noise.

2. **Sensitivity** measures by how much the *values* of the properties change with different levels of noise.

3. **Robustness** measures whether the network retains certain features as it is subjected to noise.

4. **Reliability** measures whether the network can perform certain functions, such as spread messages within a given interval, when noise is added.

As can be seen, the findings from these studies are often contradictory, if the details such as the error model or type of accuracy measured is overlooked. In our work we therefore explore perturbations at a more fine-grained level, including multiple values of \( k \) for the top-\( k \) ranked vertices and investigating the local structure of the vertex.

We now present overviews of the relevant papers for each category (stability, sensitivity, robustness, and reliability). There are cases where a paper can be classified into multiple categories and we address those cases.

4.1 Stability

In [8] the model proposed is tested on Erdős-Rényi random graphs and consists of four possible errors that can occur in the graph. These are: node removal (random removal of a proportion of the existing nodes in the graph), node addition (insertion of a proportion of extra nodes into the graph along with new edges randomly added from each of the added nodes to existing nodes), edge removal (random removal of a proportion of existing nodes), edge addition (insertion of new edges not present in the original graph). For each of the four possible errors, simulations were carried out for different graph sizes and densities, and different levels of error to see how different centrality measures were affected (degree, betweenness, closeness, and eigenvector centralities were examined). The levels of error were parameterized by percentages, then the percentage was used to determine how many nodes/edges to randomly add/remove as a fraction of the total number of
nodes/edges. To determine the effect on the centrality measures, several different metrics were used including $\text{Top}_m$ for $m = 1, 3, 10\%$ that tracked the top node for each centrality and whether it was in the top 1, top 3, and top 10% after perturbation. A measure known as "overlap" (equivalent to the Jaccard index) was also used to measure the similarity of the top 10% rankings between the original network and the perturbed versions for each centrality. The Pearson correlation coefficient was also used on the actual centrality values of the original and perturbed versions of the network, meaning this work can also be classified in the "sensitivity" category. The main result of their simulations is that the accuracy of the centrality measures in the perturbed graphs decreases with increasing error predictably and monotonically.

The stability of centrality measures is analyzed for weighted graphs in [63]. Noise is simulated as random fluctuations in the edge weights of a network using two types. In the first type of noise small changes to edge weights are done for all of the edges, and in the second type a larger amount of change is introduced to edge weights for 10% of edges selected at random. Using primarily a theoretical analysis, the authors determined that degree, closeness, and eigenvector centralities are stable while betweenness is not. A stable version of betweenness centrality is also introduced. Aside from the theoretical analysis, experiments are performed on random networks and the change in node ranking is examined for each of the measures to demonstrate the theoretical results. This work could also be categorized in the "sensitivity" category since the theoretical results are based on the actual values of node centralities rather than their ranks.

Betweenness centrality was originally modified in [62] to make it more stable and this is the measure used in [63] by the same group of researchers. It is first shown in this work that this measure preserves stability under noise using the same noise model as in [63]. The theoretical results are then demonstrated for real-world as well as random networks. This is a prime example of how centrality measures can be modified in order to demonstrate network stability specific to the customized centrality measure. This study can also be included in the "sensitivity" category.

The noise model proposed in [1] (the one used in this dissertation) is based only on edge addition and deletion. However, instead of only random changes, different noise models are used including models that are biased toward nodes with higher degree i.e. the higher degree of the node, the more likely it will edges removed/added to it. The effect of these noise models on the $k$-core is studied over a wide variety of real world networks taken from SNAP [43]. The results show that the level of overlap of the top-$k$ cores of the original and perturbed networks vary non-monotonically with the level of perturbation.

In [71], noise is modeled as missing nodes, missing edges, and false edges. The centrality measures analyzed are degree, betweenness, closeness, and eigenvector. Experiments are performed on randomly generated networks and the stability of the centrality measures is determined using the $\text{Top}_m$ and overlap (Jaccard index) metrics as in [8]. The stability for degree, betweenness, closeness, and eigenvector centralities behave
similarly for the given noise model.

The stability of rankings in complex networks as well as "super-stable" nodes are examined in [30]. Degree preserving edge rewiring is used as the noise model and experiments are performed on random as well as real-world networks. The rankings produced by Pagerank are analyzed for random networks and it is demonstrated that they are sensitive to network perturbations. It is also found that in scale-free networks, super-stable nodes appear that are not sensitive (with regard to rank) to perturbation. These super-stable nodes likely correspond to the highly ranked centrality nodes we identify when analyzing stable real-world networks.

The stability of network metrics in the presence of noise (inadequate and missing data) is analyzed in [56]. The specific problems studied include the effect on the ranking of centrality measures of paths with broken information flow and missing edges. Centralities used are: betweenness, clustering coefficient, and number of 2-paths a node is part of. The centrality measures used are shown to be stable on three different networks related to open source projects.

In [42], the stability of centrality measures is investigated for randomly generated small-world networks that are subjected to systematic error. Degree, betweenness, and closeness are studied to determine network stability under a systematic error noise model. The measures of stability used in this study were the same ones used in [8], namely $Top_m$ for $m = 1, 3, 10\%$ that tracked the top node for each centrality and whether it was in the top 1, top 3, and top 10\% after perturbation. The noise model was different however and consisted of randomly rewiring edges. The results showed that all of the centrality measures were unstable. Indeed we have also seen similar results (using the uniform edge perturbation model) for a specific kind of random network (RMAT-ER for example), though we have also demonstrated centrality ranking stability for other random networks (LFR-0.1).

Stability of centrality measures under uncertainty is also investigated in [27]. The effect of noise, termed error in this work, on various centrality measures is investigated for several different network topologies. These include: uniform random, small-world, core-periphery, scale-free, and cellular. The centrality measures analyzed include: degree, betweenness, closeness, eigenvector, and local clustering coefficient. The measures of stability for each centrality are the same as those in [8], namely $Top_m$ for $m = 1, 3, 10\%$, which were also used in [42]. The noise simulated by the random removal/addition of edges as well as nodes. It is determined that network topology is a significant factor in the stability of the centrality measures.
4.2 Sensitivity

Another model for noise simulation in networks is proposed in [82]. The noise model consists of six possible measurement errors that can occur in networks. The first four are essentially the same as the errors used in the model previously described [8] except the terminology is different. Specifically, the measurement errors are: false negative nodes/edges (node/edge removal), false positive nodes/edges (node/edge addition), and false aggregation/disaggregation. False aggregation happens when two or more nodes are erroneously classified as one node. False disaggregation happens when one node is erroneously classified as multiple separate nodes. These six types of errors were simulated on two real world networks and a random graph in order to analyze the effect on degree centrality, clustering coefficient, network constraint, and eigenvector centrality. The noise simulations are very similar to [8] in that the errors are simulated for a random selection of nodes/edges for a specified fraction of the total nodes/edges. Spearman’s rho is then used to determine how much correlation there is between the original centrality values and the values in the perturbed networks. The simulations show that networks with positively-skewed degree distributions and higher average clustering coefficient are affected more by most of the different types of error used.

The noise model in [34] consists of random node/edge removal, random node/edge addition, random node/edge removal with bias, and random node/edge addition with bias. The bias model is based on the median values of the centrality measures analyzed, nodes/edges with centrality values higher than the median are more likely to be targeted. The centrality measures used are: betweenness, closeness, and eigenvector and the experiments are performed on real-world networks. The main finding is that sensitivity of the centrality measures is dependent on the network topology.

In [13], the sensitivity of centrality measures in sampled networks are studied. Bootstrap sampling is used to determine the effect on 11 centrality measures and differentiate the measures according to sensitivity. The centrality measures analyzed are: degree, closeness, eigenvector, radiality, integration, and several variations of each centrality. Experiments are performed on real-world networks by sampling different fractions of them and then correlating the centrality distributions of the sampled networks to the original versions. The results show a high correlation between the sampled network properties and their original values. This indicates that using smaller sampled versions of a network for centrality calculations (which will take less time) can produce results that are significantly close enough to the original. Thus, if it is not possible to obtain the data for the full network, researchers can still make inferences about the whole network using an incomplete version of it.

In [80], the sensitivity of 10 different complex network measurements are analyzed on random networks as well as real-world networks. Perturbations are made on each network of the following type: edge deletion,
addition, and rewiring. The level of perturbation is determined by a percentage (0-10%) and that fraction of edges is then perturbed. Some of the 10 measures analyzed include: degree (and variations of it), assortativity, closeness, betweenness, and path length. The distributions of the various measures for the perturbed networks are compared to the original distributions in order to quantify their sensitivity. It is shown the edge deletion has less of an effect on the measurements as does rewiring, followed by addition.

Network measure stability under edge perturbations was researched in [59]. Stochastic models are used for false and missing edges in order to quantify the effect on degree, betweenness, eigenvector, and dynamical importance. The measures of stability included correlation between the original and perturbed distributions, overlap (Jaccard index) in rankings, and the Top 10% used in [8]. A theoretical analysis was also developed which agreed with the experiments. Since overlap and Top 10% measures are used to measure changes in ranking, this study can also be classed in the "stability" category.

4.3 Robustness

The effect of edge failure on the stability of biological networks as well as random and scale-free benchmark networks is investigated in [36]. The main focus is on how local network structure affects global network stability. Several measures were used to determine edge vulnerability, including a betweenness type measure (edge frequency in shortest paths in the network) which had a high correlation with edge vulnerability, and a measure based on the product/difference of degrees of edge endpoints (nodes). The noise model consisted of edge removal according to the vulnerability measures (most vulnerable edges removed first) and measuring how the network is affected in density, clustering coefficient, and average shortest paths.

The existence and sensitivity of a giant connected component (GCC) when the network is subject to noisy and incomplete observations is the topic in [69]. It is shown that both sampling and noise have a major impact on the existence/destruction of a GCC in a network. Noise in this study consists of uniform perturbations to each edge in a network i.e. each edge is removed with a certain probability and each possible edge is added with a probability, very similar to the uniform perturbation model which we use. Experiments are performed on real-world and random networks. Criteria are determined for the existence of a GCC after the network is subjected to sampling and/or noise and then demonstrated for the networks used in the experiments.

A study on how assortativity (likelihood of 2 nodes with similar degrees having an edge between them) affects the robustness of interdependent networks is presented in [87]. When networks exhibit interdependency, this can be a significant factor in amplifying failures and causing system-wide malfunction. A two interdependent network model is used to show that internal node correlations significantly change the threshold of network failures that result in complete system malfunction. The noise model used consists of the
removal of a specified percentage of nodes that are selected at random. Robustness is quantified by analyzing
the largest connected component and the networks used for the experiments are randomly generated Erdős-
Rényi graphs and scale-free graphs. Assortativity in a single network is shown to decrease the robustness of
the whole system i.e. having nodes with similar degree connected to each other will actually decrease system
robustness.

Perturbations involving vertices are studied in [35] in order to determine network robustness. Vertices are
removed according to their centrality ranking (top vertices removed first), for a variety of centrality measures
(degree, betweenness, closeness, and eigenvector). The effect of this removal is analyzed for various networks
(real-world and random scale-free) by measuring the size of the largest connected component to determine
their robustness.

Characteristics of network topologies that are robust under attack are investigated in [70]. The attacks
considered are: random removal of nodes, random removal of high degree nodes, and also removal of high
betweenness centrality/ high degree nodes. Elasticity (determined by throughput vs. percentage of node
removal) is used as the robustness measure and it is shown that different network topologies are affected
differently by the attacks. The experiments are performed on real-world networks.

The robustness of gene regulatory networks is analyzed in [54]. Various definitions and measures of
robustness (based on network dynamics, attractors, and difference in states) are used for the analysis. Noise
is modeled by the removal/addition of edges randomly. The effect that a power law distribution has on
network dynamics is investigated and it is found that the power parameter significantly contributes to the
robustness of the system, as was shown in previous studies.

The random rewiring of edges for robustness of networks is analyzed in [44]. The method presented
improves network robustness based on the largest connected component of the network. It also results in the
formation of levels of similar degree nodes, while preserving a modular structure. The strategy is tested on a
world air transportation network and produces a 30% increase in robustness.

4.4 Reliability

In [39], the Moore-Shannon reliability polynomial is used to study the effects of network structure on dif-
fusive dynamics. Structural motifs are used to represent the reliability polynomial and general results are
presented that relate network structure to network dynamics. The noise model used consisted of the random
removal of edges/nodes.

The Moore-Shannon reliability polynomial is also used in [86] to determine the reliability of random net-
works (Erdős-Rényi and scale-free) that have been manipulated to have different assortativities and numbers
of triangles. It is shown that increasing the number of triangles does not increase network reliability but positively assortative networks are actually more reliable than those with lower assortativity values. The noise model, termed attack rate in the study, consists of the removal of vertices. The largest connected component was also considered in this paper to quantify how a network is affected by noise and so this study can also be classed in the "robustness" category.

4.5 Community Detection

While the measures we focus on to determine stability in our noise experiments are different types of centrality (betweenness, closeness, degree), researchers have also examined how the community structure of a network is affected by perturbations. Communities in networks have various definitions but all of them basically define communities as groups of vertices that are connected tightly among themselves but have few connections to vertices outside of the community. Many real world networks, such as social networks, exhibit a community structure where all of the nodes can be partitioned into tightly connected groups. When the network is perturbed, the robustness of the network can be quantified by analyzing how the community structure changes in the perturbed version. We present a brief description of some of the research in this area.

The problem of making networks more robust is examined in [85]. While previous studies have focused on the degree distribution to make networks more robust, these authors propose examining the community structure. The relationship between robustness and community structure is explored and a strategy to make the network more robust is proposed. The strategy preserves community structure as well as the degree distribution and is demonstrated to improve robustness of real-world networks.

The robustness of community structure in networks is studied in [37]. While many methods of determining community structure in networks have been proposed, only a few take into account whether the network structure is significant or whether it resulted randomly. The authors in [37] show that the strength of community structure can be measured by examining its robustness to changes in the network topology.

In [29], the stability of community structure in complex networks is studied using a proposed method of identifying nodes in between communities and introducing noise to the edge weights. It is claimed that this method can be used with any network clustering algorithm to determine the stability of the community structure.

Stable communities in complex networks are shown in [64]. While there are many community detection algorithms, many of the resulting communities tend to be unstable. Stable communities also known as community cores are determined in [64] and are shown to be similar to ground truth communities.
5 Noise in Networks

Networks collected from real-world applications inherently contain some noise, i.e. false positive edges that are added, but should not be and false negative or missing edges. Sensitivity of network parameters under slight perturbations is an important concern because of this inherent noise. A network is stable if under slight changes to its connectivity, the parameters of the network do not vary significantly. We perturb the networks to see whether the ranking of the high centrality vertices is maintained under perturbation.

We used the Erdős-Rényi random graph based perturbation model developed in [1]. In this model, for a given parameter $\varepsilon$, $0 \leq \varepsilon \leq |V|$, an edge that is present in the original network has a probability of $\frac{\varepsilon}{|V|}$ of being removed, and an edge that is not part of the original network has a probability of $\frac{\varepsilon}{|V|}$ of being added. Note that if the network is fairly sparse then the perturbed network will have more edges added than removed which will result in a network with more edges than the original.

When comparing the ranking of the original network to the perturbed instances we use the Jaccard index. Given two sets $A$ and $B$, the Jaccard index is defined to be: $JI(A, B) = \frac{|A \cap B|}{|A \cup B|}$ where $|A \cap B|$ is the size of their intersection and $|A \cup B|$ is the size of their union. For two sets that are identical, the Jaccard index is 1 and for two disjoint sets the value is 0. Thus the closer the index is to 1 the lower the effect of the perturbation.

5.1 Perturbations for a Fixed Value of $k$

Using the described perturbation model, we first analyze the effects perturbations have on the ranking of the top 10 centrality vertices. We first compute the BC or CC of every vertex in each network, using the exact Brandes algorithm or the standard closeness centrality algorithm, and find the top 10 ranked vertices. Then we run the perturbation model on each network at various values of $\varepsilon$ to obtain new networks for which we again find the top 10 centrality vertices. We then compare the new set with the original set and measure their similarity using the Jaccard index.

We have observed that group testing is most effective in finding the high centrality values when the vertex rankings are maintained under perturbation. If the vertex rankings alter this indicates that the high ranked vertices are not significantly higher than their competitors and this leads to false positives in the group testing method.

5.1.1 Betweenness Centrality Analysis for Fixed $k$

Some of the networks were perturbed using 8 different $\varepsilon$ values ranging from 0.05 to 2.5. In the betweenness centrality case we computed the BC values of the vertices using the exact Brandes algorithm. We calculated the Jaccard index between the top ten BC vertices from the original set and the new sets and took the mean
Figure 10: **Effect of perturbation on the ranking of the high BC vertices.** **Left:** The ranking is stable or has a smooth decline. **Right:** Ranking is easily disrupted or has an erratic decline (except LesMis).

Table 6: **Standard deviations for perturbation on the ranking of the top 10 BC vertices.**

<table>
<thead>
<tr>
<th>Network</th>
<th>$\varepsilon = 0.05$</th>
<th>$\varepsilon = 0.1$</th>
<th>$\varepsilon = 0.25$</th>
<th>$\varepsilon = 0.5$</th>
<th>$\varepsilon = 1$</th>
<th>$\varepsilon = 1.5$</th>
<th>$\varepsilon = 2$</th>
<th>$\varepsilon = 2.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>0.0903462</td>
<td>0.111366</td>
<td>0.0714249</td>
<td>0.121228</td>
<td>0.102563</td>
<td>0.165348</td>
<td>0.117902</td>
<td>0.0796515</td>
</tr>
<tr>
<td>C. elegans</td>
<td>0.0571399</td>
<td>0.0571399</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0714249</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Chesapeake</td>
<td>0.0857099</td>
<td>0.0857099</td>
<td>0.157472</td>
<td>0.12748</td>
<td>0.115655</td>
<td>0.105118</td>
<td>0.0934018</td>
<td>0.0944907</td>
</tr>
<tr>
<td>AS1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AS2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0677596</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caida</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Les Mis</td>
<td>0.0571399</td>
<td>0.075589</td>
<td>0.124078</td>
<td>0.102132</td>
<td>0.128265</td>
<td>0.09597</td>
<td>0.0984371</td>
<td>0.107645</td>
</tr>
<tr>
<td>GrQc</td>
<td>0.0903462</td>
<td>0.0571399</td>
<td>0.132559</td>
<td>0.0752885</td>
<td>0.19029</td>
<td>0.133266</td>
<td>0.12649</td>
<td>0.0797352</td>
</tr>
<tr>
<td>HepTh</td>
<td>0.0857099</td>
<td>0.0903462</td>
<td>0.13627</td>
<td>0.0945972</td>
<td>0.0984371</td>
<td>0.0661085</td>
<td>0.0778941</td>
<td>0.106982</td>
</tr>
<tr>
<td>Power Grid</td>
<td>0.0948021</td>
<td>0.0711496</td>
<td>0.0151305</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0151305</td>
</tr>
</tbody>
</table>

of the Jaccard indices over ten runs over the same $\varepsilon$ value. As shown in Figure 10 some networks have a very stable top 10 BC set (Jaccard index very high) while some show increasing instability as the level increases. Power grid is an extreme example of instability where the top BC vertices almost never match after perturbations.

The left figure contains results for the networks on which group testing was successful. Note that the ranking is stable with high Jaccard index, or the value gradually declines. In contrast, Jaccard index values in the right figure, where group testing did not work, do not exhibit any specific pattern and even small $\varepsilon$ values have very low Jaccard index.

Table 6 shows the standard deviations for the perturbation experiments on the ranking of the top 10 BC vertices (for the results given in Figure 10). Notice that the values are all very low, in some cases (AS1, Caida) they are all 0. This means that the fluctuations in our experimental results for a fixed $\varepsilon$ are not significant enough to change our conclusions. The same is evident with the standard deviations for other centrality measures and other k-values and so we do not present them further.
5.1.2 Closeness Centrality Analysis for Fixed $k$

Figure 11 shows the perturbation results for closeness centrality in various networks. There is a clear difference between the perturbation results for betweenness centrality and closeness centrality. The effect of perturbation on closeness rankings is significantly less than the effect perturbation has on betweenness rankings and the change is more smooth as the perturbation level is increased. Also, unlike the case for betweenness, the networks for which group testing was successful are not necessarily stable under perturbation (though they have relative stability when compared to the betweenness case) and conversely those networks for which group testing was not successful are not necessarily easily disrupted by perturbation. For example Brightkite in Figure 11 appears to be one of the least stable networks when perturbed, however this network performed well with group testing.

The main reason perturbations affect betweenness centrality rankings more (for fixed $k = 10$) than closeness centrality rankings is the difference in how the two centrality measures are computed. When computing the closeness centrality for a given vertex $v$, the distances to all of the other $|V| - 1$ are calculated and so $|V| - 1$ shortest paths are considered in the computation, specifically the lengths of the shortest paths. On the other hand, the betweenness centrality of a vertex $v$ is calculated by considering all possible shortest paths between $\binom{|V|-1}{2}$ pairs of other vertices and not the actual distances. Since there can be multiple shortest paths between a pair of vertices, the number of shortest paths considered in the computation of betweenness is far greater than the number of shortest paths considered in closeness centrality computation. Thus, when the network is perturbed, a greater number of shortest paths are affected in betweenness centrality than closeness centrality and hence the drastic difference in how the rankings are affected.
5.2 Perturbation Analysis for a Range of $k$ Values

We continued the analysis of how the networks behave under noise for a range of $k$ values. Most of the following results are part of our paper which has been submitted to the ACM International Conference on Information and Knowledge Management (CIKM 2016) [78].

Based on our experiments on real-world networks we observe two important features that affect the stability of the networks. First, the stability is dependent on the value of $k$, i.e., the number of top-ranked vertices considered. When the vertices are ordered according to their centrality values, they group into clusters. Within the clusters, the centrality values of the vertices are very close. Perturbations to the network can change the relative ranking within the cluster, but vertices rarely move from one cluster to another. Thus if the value of $k$ falls in the middle of a cluster, the stability is low. If $k$ is at the beginning of a cluster stability is high.

Second, the stability is dependent on the local connections of the high ranking vertices. The network is highly stable if the high ranking vertices are connected to each other and is less stable when high ranking vertices do not induce a dense neighborhood (aka rich-club in the literature).

Our findings show for the first time that the stability of a network under noise is affected by the local properties of high centrality vertices. This is an important departure from earlier studies that looked only at global properties to understand network stability. Based on these local properties we can now identify whether a network is stable or not, without explicitly applying a noise model.

In the following experiments we used the additive uniform perturbation model, where a percentage of edges, selected from the complementary graph in random, are added to the existing network.

This type of noise occurs when creating networks from correlation values of raw data. Here the raw data shows how each data point changes with respect to a parameter. In biology, this can be microarray studies of how a gene reacts to different stimuli. In environmental studies, this can be how the temperature or the precipitation of a place changes each week. Two data points (vertices) are connected, if their array of parameters have high correlation values. Note that as the cut-off for high correlation is decreased, the network gets progressively denser, i.e. more edges are added.

In social networks, it is claimed that noise can be both additive (edges added) and reductive (edges deleted), although surveys in general tend to bias more on one direction. Moreover missing edges are easier to fix using the different link prediction algorithms available. To the best of our knowledge, there is no such algorithm to specifically identify the extra edges that were added erroneously. Therefore understanding the effect of extraneous edges is the more critical problem, and that is what we focus on in the following experiments.
In addition to analyzing how betweenness and closeness centralities are affected by noise, as we have presented earlier, we also analyze how degree centrality (number of immediate neighbors) is affected.

For degree, betweenness and closeness centrality we computed the stability of each network up to $k=10$. If we take larger values of $k$ then the ranking is not as clear because the centrality values become very similar. We therefore look at the stability within the range where the centrality values are distinct. As before, we compute the Jaccard index to test how many of the top $k$ vertices in the original network are also among the top $k$ vertices in the perturbed network.

We conducted these experiments for each network over a set of 10 perturbed networks per noise level. The Jaccard index presented in the results is the mean over the 10 networks. The standard deviation was low, ranging from 0 to 0.14. Based on the value of the Jaccard index (JI), we classified the stability into three groups as follows; High Stability ($JI \geq 0.7$); Medium Stability ($0.4 \leq JI < 0.7$) and Low Stability ($0.4 > JI \geq 0$).

Table 7 shows the networks used in the perturbation analysis for a range of $k$-values.

Table 7: Test suite of real-world networks and their properties. The degree distribution slope $\alpha$ is mostly within 1.5 to 3.00 and the global clustering coefficient $CC_f$ has a wide spectrum from .10 to .74.

<table>
<thead>
<tr>
<th>Name</th>
<th>Nodes</th>
<th>Edges</th>
<th>$CC_f$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS2</td>
<td>6474</td>
<td>13895</td>
<td>0.39</td>
<td>1.49</td>
</tr>
<tr>
<td>AS1</td>
<td>3570</td>
<td>7750</td>
<td>0.31</td>
<td>1.57</td>
</tr>
<tr>
<td>C. elegans</td>
<td>453</td>
<td>2025</td>
<td>0.65</td>
<td>1.65</td>
</tr>
<tr>
<td>Les Mis.</td>
<td>77</td>
<td>254</td>
<td>0.73</td>
<td>3.05</td>
</tr>
<tr>
<td>GrQc</td>
<td>5242</td>
<td>14496</td>
<td>0.68</td>
<td>1.78</td>
</tr>
<tr>
<td>HepTh</td>
<td>9877</td>
<td>25998</td>
<td>0.59</td>
<td>1.66</td>
</tr>
<tr>
<td>Power Grid</td>
<td>4941</td>
<td>6594</td>
<td>0.10</td>
<td>1.45</td>
</tr>
<tr>
<td>Railway</td>
<td>301</td>
<td>1224</td>
<td>0.74</td>
<td>6.68</td>
</tr>
<tr>
<td>Football</td>
<td>115</td>
<td>613</td>
<td>0.40</td>
<td>1.57</td>
</tr>
<tr>
<td>Email</td>
<td>1133</td>
<td>5451</td>
<td>0.25</td>
<td>2.75</td>
</tr>
<tr>
<td>Dolphins</td>
<td>62</td>
<td>159</td>
<td>0.30</td>
<td>5.53</td>
</tr>
</tbody>
</table>

Figure 12: Changes in dominant stability of centrality metrics over different noise levels. Left: Degree Centrality, Middle: Closeness Centrality, Right: Betweenness Centrality. X-axis: Noise Levels. Y-axis: Number of times the stability value fell in the high (H), medium (M) or low (L) range. The dominant stability decreases with increased noise level, but the rate of decrease depends on the centrality metric.

We provide a summary of how the stability of the networks change with the noise levels in Figure 12. We
also report the stability of individual networks changed for different noise levels, different values of \( k \), i.e. the number of top vertices considered and different centrality metrics in Figure 13.

In the summary figure (Figure 12), each line represents a network. The X-axis represents the different noise levels (\( \epsilon \)). For ease of visualization here we plot only for the even values of \( k \) from 2 to 10. In the Y-axis we measure the dominant stability, i.e. the longest consecutively occurring stability range for that noise level. For example, 5H denotes that at that noise level, for all the five values of \( k \) the stability was high. Similarly, 3M denotes that for three consecutive values of \( k \) the stability was in the middle range.

Figure 13 shows the changes for the individual networks, per value of \( k \), not just the dominant stability. We have included two networks that were consistently in the high range (AS1 and C. elegans), two that were consistently in the low range (Power Grid and Football) and two that changed their stability values according to the centrality metric and noise level (GrQc and Railway). The results show that the stability value can change depending on the value of \( k \). More detailed are charts for more values of \( k \) and a wider range of noise levels are presented in Appendix B.

The results show that even a small amount of noise (average edges added per vertex is 2.5) can significantly change the analysis results. However, the behavior of the three centrality metrics varies as follows:

**Degree:** The dominant stability decreases monotonically for degree centrality. With the exception of Power Grid (has some middle level stability) and Football (all stabilities are low) all other networks show high stability.

**Closeness:** For closeness centrality there are several networks that show predominantly low dominant stability (Power Grid, Football, GrQc and Dolphins). Networks HepTh and Railway start as high stability, but their stability decreases with higher noise levels.

**Betweenness:** A similar division can be observed for betweenness. Here, Power Grid, Football and Dolphins have low dominant stability. GrQc goes from high, to medium to low. Railway also starts from high and ends at low.

The trends are similar when we see the changes in stability with respect to values of \( k \). However, the division is not as clear. Depending on the value of \( k \), the stability can change from high to low and then back to high. To summarize, our main observations from these results are as follows;

- The dominant stability decreases with increasing levels of noise. However, the individual stability changes non-monotonically with the values of \( k \).

- The stability of the network depends on the centrality metric. Degree is most stable, closeness has a clearer separation between the high and low stability networks and in betweenness the separation is not as clear. The same network (e.g. GrQc at noise level 1.5) can have high (degree), low (closeness) and
Figure 13: Change in centrality values for different networks and various choices of $k$, over different noise levels. Left: Betweenness, Middle: Closeness, Right: Degree. Y-axis: Jaccard Index. X-axis: Value of $k$, the number of top vertices considered. The first two rows are networks with high stability, the middle two are networks with low stability and the last two are networks whose betweenness and closeness stability changes from high to low and back. The choice of $k$ can significantly affect the JI value.
medium (betweenness) stability based on the centrality metrics.

- The global topology of the network is not a deciding factor. As per Table 4, the clustering coefficients of the networks are very diverse and the slope of the degree distribution curve is between 1.5 and 3. However, unlike the conclusions of [82] neither of these parameters seem to strongly correlate with the stability values.

5.3 Stability of Centrality Metrics

Figure 14: Stable clusters of centrality values Top Graphs: Betweenness Centrality. Bottom Graphs: Closeness Centrality. Line Graphs: X-axis plots \( k \), the number of top centrality vertices considered. Y-axis plots the Jaccard Index. These graphs show how the stability changes with different values of \( k \), as noise levels remain constant. Scattered Plot: X-axis plots vertex id for the top 10 centrality vertices. Y-axis plots the centrality values. The vertices can be clustered based on the relative difference of their centrality values. By comparing each pair of line graph and scattered plot, we see that in general the stability increases when \( k \) is in the beginning of a cluster and decreases when \( k \) falls within the cluster.

In this section we further explain how properties of the network affect the stability. We first look at how likely the order of the top-\( k \) nodes of the centrality metrics is to change under the noise model we have been implementing. We show that there is a critical difference of values between consecutively ranked vertices after which the order will remain stable. Therefore, we next look at the distribution of the centrality values and demonstrate that indeed the vertices can be clustered based on their centrality values. Based on these observations, we conjecture that the local properties of high ranking vertices determine the stability of the system and present results to support this claim.
5.3.1 Theoretical Analysis of Centrality Stability

Consider two nodes $v_1$ and $v_2$, whose values for a particular centrality metric, are $X(v_1)$ and $X(v_2)$ respectively. In the original networks let $X(v_1) > X(v_2)$, therefore $v_1$ has a higher rank than $v_2$. After applying a particular perturbation $p$, the centrality values of $v_1$ and $v_2$ become $X_p(v_1)$ and $X_p(v_2)$.

Our goal is to identify the lower bound on the difference between $X(v_1)$ and $X(v_2)$, such that after perturbation $X_p(v_1)$ will still remain greater than $X_p(v_2)$. To compute this difference, we consider the most optimal situation for $X_p(v_2)$ to become larger than $X_p(v_1)$. We assume that $X(v_1)$ has the maximum decrease after perturbation and $X(v_2)$ has the maximum increase, given that on average $\varepsilon$ edges are added per vertex. Our computations for each centrality values are as follows;

**Degree centrality:** The change in degree of a vertex is equal to the number of edges added to it. Therefore, the degree of a vertex will either increase or remain the same. Thus the maximum decrease of $d$ is zero, i.e. $X(V_1) = X_p(v_1)$. The value of $X_p(v_2) = X(v_2) + \varepsilon$. Therefore, if $X(v_1) - X(v_2) > \varepsilon$, then the ranking will not change.

For most networks the difference between the higher ranked vertices is larger than the maximum $\varepsilon$ we set for our experiments, so the ranking of the vertices remain relatively stable.

**Closeness centrality:** For simplicity of calculations, we consider $X(v)$ to be the inverse of closeness centrality, i.e. $X(v) = \sum_{s \neq v \in V} dis(v,s)$

Since we are adding edges, this value will either increase or remain the same. Thus once again $X(V_1) = X_p(v_1)$. The change in $X(v_2)$ will depend on where the edges are added.

Assume, due to perturbations, $v_2$ is added to a vertex $v_x$, which is at distance $d_x$ from $v_2$. Therefore, $v_x$, and other vertices whose shortest paths to $v_2$ passed through $v_x$ will have their distance to $v_2$ reduced by $d_x - 1$. The maximum decrease is $X_p(v_2) = X(v_2) - \sum_{t \in E_{add}} (d_t - 1)R_t$, where $E_{add}$ is the set of nodes that are added to $v_2$, $d_t$ is distance of $t$ from $v_2$ in the original graph and $R_t$ is the number of vertices whose shortest path to $v_2$ passes through $t$. Thus the following has to hold: $X(v_2) - X(v_1) > \sum_{t \in E_{add}} (d_t - 1)R_t$ for the ordering between these two vertices to be stable. $R_t$ will increase with $\varepsilon$. The values of $d_t$ depends on the depth of the BFS tree originating from $v_2$.

**Betweenness Centrality:** By adding edges the betweenness centrality of a vertex can increase if it gets connected to another high centrality vertex. However, the BC can also decrease, if new edges to other vertices combine to create alternate or smaller shortest paths.

Assume due to addition of edges to a vertex $v$, there are $R$ new pairs of vertices whose shortest paths pass through $v$. Also due to addition of edges in other parts of the networks, there are $P$ pairs of vertices whose

---

1We consider ranking from 1 (high) to $n$ (low). The vertex with highest centrality value is ranked 1
shortest paths used to pass through \( v \) in the original network, but do not in the perturbed network. There are also \( Q \) pair of edges, whose length of shortest path does not change, but after perturbation there are new shortest paths between them.

We assume that \( v_1 \) sees only decrease in its BC value and \( v_2 \) sees only increase. Therefore

\[
X_p(V_1) = X(v_1) - \sum_{s_p \neq v_1 \neq t_p \in P}^{s_p \neq v_1 \neq t_q \in Q} q_e(\sigma_{sptp}(v_1)) - \sum_{s_q \neq v_1 \neq t_q \in Q}^{s_q \neq v_1 \neq t_q \in Q} q_e(\sigma_{sqtq}(v_1)) - \sum_{s_r \neq v_2 \neq t_r \in R}^{s_r \neq v_2 \neq t_r \in R} \sigma_{srtr}(v_2),
\]

where \( \sigma \) is the number of new shortest paths for the vertex pair \( s_q \) and \( t_q \) and \( \sigma_{sqtq}(v_1) \) is the number of new shortest paths for the vertex pair \( s_q \) and \( t_q \). Therefore, the difference between \( X(v_1) - X(v_2) \), must be larger than

\[
\sum_{s_p \neq v_1 \neq t_p \in P}^{s_p \neq v_1 \neq t_q \in Q} \sigma_{sptp}(v_1) + \sum_{s_q \neq v_1 \neq t_q \in Q}^{s_q \neq v_1 \neq t_q \in Q} q_e(\sigma_{sqtq}(v_1)) + \sum_{s_r \neq v_2 \neq t_r \in R}^{s_r \neq v_2 \neq t_r \in R} \sigma_{srtr}(v_2).
\]

The number of elements in \( R \) will increase as \( \varepsilon \) increases. The number of elements in \( P \) and \( Q \) depend on the length of the shortest paths. If the length of most of the shortest paths through \( v_1 \) is already low, then there is less chance that they will become even shorter or alternate paths will be found with addition of new edges.

In summary, increasing \( \varepsilon \) leads to increase in the centrality values, which can lead to re-ordering of vertex ranks. This is why, as shown in Figure 12, the stability tends to decrease with higher \( \varepsilon \). However, for closeness and betweenness centrality, the increases also depend on network structure.

5.3.2 Stability Based on the Difference in Centrality Values

From the previous discussion we see that if the difference in centrality between vertices in the original network is more than a certain value, then the relative order of the vertices will likely not alter under perturbation.

However, apart from degree, this lower bound on the difference in values depends on the network structure, and the probability that certain special vertices will get connected. However, before we consider the network structure, we observe that simply looking at the relative differences of consecutive centrality values can indicate whether the ordering will be maintained or not. Figure 14 plots the change in stability as the noise levels remain constant, and the value of \( k \) changes (line-graphs) and the values of the top-10 high centrality vertices (scattered plots).\(^2\) The plots show that the vertices can be grouped into clusters, where within the clusters the values are relatively close to each other, and across the clusters there is a large difference between the last vertex in the previous cluster and the first vertex in the next cluster. The stability increases when \( k \) is at the beginning of a cluster (e.g. \( k = 4 \) for AS1 Betweenness Centrality) and decreases when \( k \) falls within the cluster (e.g. \( k = 3 \) for AS1 Betweenness Centrality).

**Stable Clusters:** This phenomena can be easily explained by considering that it is more difficult to reverse

\(^2\)The rank 1 node is not shown since its value is very high. By plotting it, the relative difference between the rest of the vertices cannot be visualized well.
Figure 15: **High ranked common neighbors of the top-k ranked vertices.** The left-hand set of figures are for $k = 10$ and the right-hand set are for $k = 6$. The top graphs are for real world networks and the bottom ones for synthetic networks. X-axis is the number of high ranked neighbors considered. Y-axis is the average JI, indicating the number of common neighbors among the top-$k$ vertices. For the high stability networks, the slope is increasing. This indicates that in stable networks, the most of the common neighbors of high ranked vertices are other high-ranked vertices.

the relative ranking between two vertices if they have a large difference in their values. However if the values are very close then slight perturbation can change the rankings.

Therefore, even without considering the network structure, we can use the relative difference between consecutively ranked vertices to group similarly valued vertices into clusters. We term these clusters as **stable clusters**. Within a stable cluster the ranks can change under slight perturbation to the network, therefore if the value of $k$ falls within the cluster the Jaccard Index is likely to change. On the other hand, if $k$ is selected such that it falls at the beginning of the cluster, then the ranking becomes more stable due to the large relative difference. This observation is borne out in Figure 14.

**Identifying Stable Clusters:** We can identify the stable groups as follows. We compare the difference between the centrality values of the consecutively ordered vertices. The first break into clusters occurs between the two vertices that have the highest relative difference. The second break between the vertices having the second highest difference, and so on. We continue dividing the vertices into clusters until the difference is lower than a certain threshold. At this point we have obtained the stable groups. The clusters in Figure 14 have been identified using this method.

Identifying these stable clusters allows us to have a improved understanding of how the network will behave under various levels of noise. **Networks where the clusters are small in size and the clusters have high difference between them should have high stability.**
5.3.3 Stability Based on Network Structure

We now consider how the structure of the network affects its stability. As seen earlier, the slope of the degree distributions ($\alpha$) for most of the networks in our test suite are from 1.5 to 3, their average local clustering coefficient is very varied, and neither of these factors correlate to the stability of the networks. For example AS2 and Football have similar $\alpha$ values (1.49 and 1.57 respectively), as well as similar global clustering coefficient (.39 and .40). Yet AS2 is a highly stable network while football has very low stability. Therefore, as seen from the theoretical analysis, the stability seems to be dependent on the local structure of the high centrality vertices instead of the global structure.

High ranked common neighbors: Note that vertices will have high closeness and betweenness centrality, if they are also connected to other high centrality vertices. Based on this observation, we investigated whether the common neighbors of the top $k$ high ranked nodes also have high rank. For each pair of nodes within the top $k$ set (for a given centrality metric, and network) we calculated the Jaccard Index between their connections to the top 100, top 50, top 25, and top 10 high ranking nodes, and computed the average JI for each set of neighbors (top 100, top 50, etc.). We conducted these experiments for $k = 10$ and $k = 6$.

As shown in Figure 15, as the range of high ranked neighbors decreases (from 100 down to 10), the average JI value increases. This indicates that the top-$k$ high rank nodes have more common neighbors among the high-ranked nodes. Also observe that the curves neatly divide into three regions. The top networks are the ones with high stability (e.g., C. elegans), the networks in the middle are ones with not so high stability (e.g., Email) and the ones at the bottom show low stability (e.g., Football).

Subgraph induced by high ranked vertices: This result motivated us to see if the high ranked vertices are...
Table 8: Comparing Stability and Local Connections of Networks. **Network Stability** gives the mean of the stability over the noise levels at the specified $k$. **Subgraph Density** gives the density of the subgraph induced by the top $k$ vertices. **Common Top Neighbors** reports whether the corresponding line in Figure 15 was in high, medium or low range. **Networks with dense clusters of high ranking nodes are highly stable and networks with sparse clusters of high ranking nodes have low stability.**

<table>
<thead>
<tr>
<th>Network</th>
<th>Network Stability</th>
<th>Subgraph Density</th>
<th>Common Top Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Closeness</td>
<td>Betweenness</td>
<td>Closeness</td>
</tr>
<tr>
<td><strong>Top 10 High Ranked Vertices</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Cluster and High Stability Networks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AS20000101</td>
<td>High (.96)</td>
<td>High (1)</td>
<td>.97</td>
</tr>
<tr>
<td>AS20000102</td>
<td>High (1)</td>
<td>High (.78)</td>
<td>.95</td>
</tr>
<tr>
<td>C. elegans</td>
<td>High (.94)</td>
<td>High (.76)</td>
<td>.82</td>
</tr>
<tr>
<td>Les Mis</td>
<td>High (.8)</td>
<td>High (.76)</td>
<td>.66</td>
</tr>
<tr>
<td><strong>Sparse Cluster and Low Stability Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrQc</td>
<td>Low (.26)</td>
<td>Medium (.64)</td>
<td>.26</td>
</tr>
<tr>
<td>Dolphin</td>
<td>Low (.1)</td>
<td>Low (.1)</td>
<td>.36</td>
</tr>
<tr>
<td>Football</td>
<td>Low (0)</td>
<td>Low (0)</td>
<td>.16</td>
</tr>
<tr>
<td>Power Grid</td>
<td>Low (0)</td>
<td>Low (0)</td>
<td>.24</td>
</tr>
<tr>
<td><strong>Outlier Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td>High (.98)</td>
<td>High (.96)</td>
<td>.31</td>
</tr>
<tr>
<td>Railway</td>
<td>Medium (.68)</td>
<td>Medium (.68)</td>
<td>.67</td>
</tr>
<tr>
<td>HepTh</td>
<td>Medium (.68)</td>
<td>High (.72)</td>
<td>.17</td>
</tr>
<tr>
<td><strong>Synthetic Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFR5000</td>
<td>High (.78)</td>
<td>High (1)</td>
<td>.77</td>
</tr>
<tr>
<td>RMAT12</td>
<td>Medium (.58)</td>
<td>Medium (.48)</td>
<td>.04</td>
</tr>
<tr>
<td><strong>Top 6 High Ranked Vertices</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Cluster and High Stability Networks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AS20000101</td>
<td>High (.86)</td>
<td>High (1)</td>
<td>1</td>
</tr>
<tr>
<td>AS20000102</td>
<td>High (.8)</td>
<td>High (.96)</td>
<td>1</td>
</tr>
<tr>
<td>C. elegans</td>
<td>High (.96)</td>
<td>High (1)</td>
<td>1</td>
</tr>
<tr>
<td>Les Mis</td>
<td>High (.90)</td>
<td>High (1)</td>
<td>.87</td>
</tr>
<tr>
<td><strong>Sparse Cluster and Low Stability Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrQc</td>
<td>Low (.22)</td>
<td>Medium (.60)</td>
<td>.20</td>
</tr>
<tr>
<td>Dolphin</td>
<td>Low (.12)</td>
<td>Low (0)</td>
<td>.47</td>
</tr>
<tr>
<td>Football</td>
<td>Low (0)</td>
<td>Low (.10)</td>
<td>.27</td>
</tr>
<tr>
<td>Power Grid</td>
<td>Low (0)</td>
<td>Low (0)</td>
<td>.27</td>
</tr>
<tr>
<td><strong>Outlier Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td>High (.90)</td>
<td>High (.76)</td>
<td>.33</td>
</tr>
<tr>
<td>Railway</td>
<td>Medium (.68)</td>
<td>Medium (.62)</td>
<td>.73</td>
</tr>
<tr>
<td>HepTh</td>
<td>Medium (.66)</td>
<td>High (1)</td>
<td>.27</td>
</tr>
<tr>
<td><strong>Synthetic Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFR5000</td>
<td>High (.72)</td>
<td>High (.84)</td>
<td>.73</td>
</tr>
<tr>
<td>RMAT12</td>
<td>Medium (.64)</td>
<td>Medium (.6)</td>
<td>.07</td>
</tr>
</tbody>
</table>

themselves tightly connected with each other. For each metric, we identified the top $k$ high ranked vertices and then computed the density of the induced subgraphs from the vertices in this set. **Networks that achieve more instances of high stability have more dense subgraphs.** The difference among the three classes of networks in terms of their subgraph structures is shown in Figure 16. While the highly stable networks such
Figure 17: **Stability of centrality metrics for synthetic networks.** X-axis: Values of \( k \) Y-axis: JI for different noise levels. **Left:** Scale-Free LFR graphs **Right:** Random graphs. **Scale-free graphs with strong community structure are more stable than random graphs**

as C. elegans induce a very dense subgraph, the networks that are less stable like Football induce an extremely sparse subgraph. For networks that are medium stable like Railway, the density of the subgraph is of medium density.

Table 8 summarizes the experimental results from this section for \( k = 10 \) and \( k = 6 \). The density of a subgraph is computed by the ratio of the total number of edges in the subgraph by the total possible edges if all the vertices in the subgraph were connected. Therefore the highest density possible is 1 (all vertices are connected) and lowest is 0 (none of the vertices are connected).

In general, if the networks have high stability for the top-\( k \) vertices, then the subgraph induced by those vertices is also dense (\( \geq .60 \)). Conversely, if the network has low stability, then the corresponding subgraphs are sparse (\( \leq .40 \), with Dolphin being the exception). This pattern is also observed when comparing their
common neighbors. For high (low) stability networks, the corresponding line in Figure 15 is in the high (low) range. We also note that the results are very similar for \( k = 10 \) and \( k = 6 \).

The exceptions to this rule are tabulated under Outliers. For example, Email and HepTh have high stability but low density of the subgraph. In these cases, we have observed that a smaller subgroup of the high centrality vertices form a dense cluster, and the rest of the high centrality vertices connect to that cluster. In some cases, there might be multiple clusters. For example, Railway has medium, tending to high, stability for betweenness. In Figure 16, we see that the subgraph for betweenness consists of two smaller clusters connected to each other. Similar characteristics can be observed for Les Mis and GrQc (high BC, low density).

**Template to Detect High Stability Networks** From our experiments we see that the stability depends on the type of centrality metric studied, how many top neighbors were considered, as well as the network structure. Based on these observations we propose a template to identify stable networks as follows:

1. Identify the top-\( k \) centrality nodes and compute their values

2. **Stability Condition 1:** Identify the lower bound between the differences of the centrality values that will maintain the ordering. If the difference in the high centrality nodes is greater than the lower bound, then the network is stable for that range of \( k \).

3. **Stability Condition 2:** Find stable clusters based on the values of high centrality nodes. If \( k \) falls at the beginning of the cluster, then the network is stable for that range of \( k \).

4. **Stability Condition 3:** Find the subgraph induced by the top-\( k \) nodes and how their number of common high ranked neighbors increase. If the subgraphs are dense and the number of common high ranked neighbors is high, then the network is stable for that range of \( k \).

Some of these conditions, such as Condition 1 for closeness and betweenness, have to be approximated, and since all these conditions depend on the network structure, they are also interrelated with each other. Nevertheless, if all these conditions are satisfied, the network should be highly stable. Conversely, if none of these conditions are satisfied the network should have low stability. Note that our method does not require the user to actually perturb the network to estimate its stability.

From our experiments we see that we get false negatives but rarely false positives when classifying high stability networks. The reverse (false positives but rarely false negatives) occurs when classifying low stability networks. This is because separation between the high, medium and low stability can depend on the different thresholds selected.

**Synthetic Networks - RMAT and LFR**
Most of the real-world networks available from the repositories have similar slope of the degree distribution curve or in some cases grid-like. e.g. Power Grid. In order to compare how stability is influenced by the overall degree distribution, we experimented with a large-scale random network (generated using RMAT [10] software; with \( a = b = c = d = .25 \), and average degree 8) and a scale-free network with strong communities (using LFR [41] with \( \mu = .1 \) and other parameters as given in default).

We applied the perturbations on several synthetic networks (vertices range between 100-10K). Figure 17 shows the results for degree, closeness and betweenness centrality for RMAT with \( 2^{12} = 4096 \) vertices and 32K edges (RMAT12), and LFR with 5K vertices and 40K edges (LFR5000).

As shown in the plots, LFR is more stable than RMAT for all the centrality values. In particular LFR shows perfect stability for degree for all the noise levels. Also note that RMAT12 has medium, not low stability.

In Table 8, shows the subgraph densities for the RMAT12 and LFR5000 for top ranked vertices as per different centrality metrics. Observe that for closeness and betweenness centralities the subgraph density of RMAT12 is constant, as should be expected from a random graph. LFR5000 however, shows strong subgraphs for closeness and has a strong cluster over a subset of vertices for betweenness centrality.

These results highlight that overall degree distribution also plays a key role in stability. When compared to random graphs, scale-free networks with strong communities are more stable.
6 Software and Applications

In this section we present a description of the software we have been developing for network analysis (ESSENS) as well as some of the applications to bioinformatics we have worked on using ESSENS.

6.1 ESSENS

We have been developing a software package, ESSENS (Extensible, Scalable Software for Evolving Networks) [76], that can be easily used to implement our algorithms on large networks.

There exist multiple libraries for both general graph operations and network analysis algorithms (as mentioned in the introduction), but there is no components based standardization of both of these key sets of operations. Therefore we propose a framework that abstracts the data structures, architecture, and programming models for the graph algorithms underneath a very simple component based interface.

We design key building blocks, which will be the basis for the software, by examining the fundamental definition of graphs. By definition, graphs are characterized by two sets: a set of elements corresponding to vertices (along with their properties) and a set of edges corresponding to relations between the elements. Generally, graph algorithms find other meaningful relations and properties based on these two sets. Using this set-based view, we propose the following building blocks that can serve as the basis for graph algorithms:

- Graphs: Set of vertices (with or without properties); Set of edges (with or without properties).
- Set operations: Set/Sequence operations on lists of vertices or edges. The set of operations are; Intersection, Difference, Union (or Merge), Subset (identify subsets that follow certain property; equivalent to filtering), Sort (including Priority Queue operations) and Find.
- Traversals: Traversals are equivalent to finding transitive chains. We start from a set of elements (generally the set is a singleton or the vertex marked as 0 and based on certain relations, continue to find transitive chains until a stopping condition (such as based on number of elements visited, or the length of the chain). Different traversals are distinguished only by which relationship (here we term it priority) drives the chain formation.
- Output: A set of elements and properties; A set of relations (edges); A scalar value.

Graph algorithms in ESSENS are classified into three levels based on area of operations. Most combinatorial methods consist of a combination of these three levels, thus this classification enables us to design and analyze our methods over a generalized framework. The three levels are:
• Level 1 - Vertex Based Computations. Computations involving only a vertex and its distance-k neighbors, where k is small. These operations are generally the least expensive ones in the analysis process. Examples include computing degree and clustering coefficient.

• Level 2 - Subgraph Based Computations. These computations involve a specified set of vertices. One example is combining certain groups of vertices in a supernode.

• Level 3 - Graph Based Computations. These computations involve traversing the entire network. Examples include verifying connectivity, finding articulation points, computing betweenness centrality.

We implemented the Brandes algorithm [9] for computing betweenness centrality using the Level 1 operations in ESSENS. In this case, we do not use any generalized functions and the runtime of ESSENS is considerably lower than that of the algorithm provided by Boost Graph Library as shown in Figure 19.

ESSENS is written in STL/C++ and is currently for undirected networks. As can be seen in Figure 18, ESSENS has a bottom-up design. The lower rows in the figure contain functions that are used in the...
upper rows. The top level shows abstractions for network-based algorithms, including: a network transform component, a computing metrics component, and a rank and compare component. Analysis methods can include several different components. For instance, finding the minimum weight spanning tree requires both network transform (spanning tree) and rank and compare (edge weights). The second level shows graph abstractions that are required by top level algorithms including: traversal of networks, subgraph operations, and matrix operations. The third level shows vertex-level functions for adding, deleting and selecting vertices and edges and auxiliary algorithms for sorting and set operations.

These levels are built on top of an implementation framework; network bundle. Users can choose to implement their own data structure for storing networks, in ESSENS the current options are adjacency list and CSR format. ESSENS will also have capabilities for supporting external packages on database storage and parallel computing.

6.2 Bioinformatics

We have used a parallel implementation of ESSENS for filtering biological networks [15]. Data intensive biological experiments are crucial in systems biology, for example in understanding cellular mechanisms and what happens in disease states. The output data from such experiments is huge in size, heterogeneous, and requires sophisticated computational algorithms for analysis. Correlation networks have been used previously for modeling and analysis of such massive data and using graph theory, structures were identified that correspond to key players in major cellular pathways. Filtering such networks i.e. reducing them in size while preserving the main structure of the network, has been shown to reduce noise and strengthen biological signals. The process of filtering biological networks is expensive computationally and thus we developed a parallel template to significantly reduce the computation time.

We have also performed edge perturbation analysis on biological networks. Using the same noise model that we implemented previously (uniform perturbation model) we performed similar experiments on several related biological networks in order to determine their stability. Once again it was evident that degree centrality was the most stable for these networks followed by closeness centrality. The biological networks were also much more sensitive to noise than the real-world and random networks we used for our main experiments. We had to significantly reduce the noise level (to $0.005 \leq \epsilon \leq 0.02$) for closeness centrality since the rankings were completely disrupted at the noise levels we previously used.
7 Future Work

We now present some the future directions of our research.

7.1 Iterative Method to Find More High Centrality Vertices

The current group testing method we are using, which is based on Latin squares, guarantees theoretically that we will identify two highly ranked vertices. We would like to identify the next highest ranked vertices also and so we propose an iterative method that will identify more high ranked vertices. We are still developing the algorithm thus we have not obtained results yet however preliminary implementations of some of the steps in the method show promise. We now provide a brief overview of the algorithm.

The process begins with running group testing on a network to identify some high centrality nodes. The identified nodes are removed from the original network and the network is updated in order to preserve the original centrality values of the remaining nodes. For closeness centrality this is done in the following way. Suppose node \( v \) was identified to have high closeness centrality. The node \( v \) is removed from the network and the edges between neighbors are updated so that distances to all of other nodes are preserved. This is done in a straightforward way.

Suppose \( u \) and \( s \) are neighbors of \( v \). If there is no edge between \( u \) and \( s \), then an edge is added with weight equal to the sum of the weights of the edge between \( u \) and \( v \) and \( s \) and \( v \). That is, \((u, s)\) is added as an edge and \( w(u, s) = w(u, v) + w(s, v) \). This preserves the distance between \( u \) and \( s \) through the node \( v \) (which will be removed). On the other hand, if there is an existing edge between \( u \) and \( s \), then the weight of it is updated as follows: \( w_{new}(u, s) = \min(w(u, s), w(u, v) + w(s, v)) \), thus again preserving the distance between \( u \) and \( s \). If it shorter through \( v \) (which will be removed) then the weight is updated otherwise it remains the same.

Once all of the edges between neighbors have been updated, the existing edges from \( v \) to the neighbors are removed and \( v \) is also removed. A set of values has to be saved in order to have the centrality values remain exactly the same numerically. These are the distances from \( v \) to all the other vertices (one BFS, \(|V| - 1\) values). After the removal of \( v \), when calculating the closeness of another vertex, the distance to \( v \) when \( v \) was part of the network has to be included in the sum of the distances or the value will not be the same.

Once the removal process has been completed for the nodes identified as having high closeness centrality, the group testing algorithm is run again on the new network to identify the next highest nodes. These nodes will also be removed, preserving centrality values, and group testing will be performed again and so forth. This iterative process of doing group testing and removing identified nodes while preserving the other centrality values is repeated until group testing fails i.e. identifies too many false positive nodes.

The central idea in the method is that after each iteration of group testing and removal of the identified
nodes, the next highest centrality nodes are actually the highest in the modified network since centrality values are preserved. As long as the next highest nodes are also significantly higher than the others then we can use group testing to identify them also. At some point, the highest nodes will no longer be very distinguished in value from the rest of the nodes, and this is exactly when group testing will fail. This signifies that the remaining nodes in the network actually do not have much importance according to the closeness centrality measure and therefore can be disregarded.

We plan to fully implement this method and experiment with identifying a greater number of high centrality nodes than if we were to perform group testing just once.

### 7.2 More Advanced Group Testing Designs

Although our current method focuses on finding only the top two centrality vertices, there exists other superimposed codes, such as those constructed from Reed-Solomon codes and randomly generated superimposed codes, that potentially could be used to find \(d\) top centrality vertices for a specified value of \(d\). We plan to experiment with other strategies to determine if they can be used on networks to identify larger numbers of highly ranked vertices. Alternatively we can also use the Latin Square method iteratively by removing the identified high centrality vertices at each iteration (while preserving other centrality values) then identify the next highest and so forth. Other group testing designs that I have worked prior to applying group testing to networks are presented in Appendix A. Although we have not yet implemented those designs for identifying high centrality vertices in networks, they could more successful than the Latin square method on networks. Appendix A also contains results on bounds for group testing (derived in 2006) which have still to be published and we hope to do so soon.

### 7.3 Alternative Network Noise Models

While we have used the uniform perturbation model from [1], there are many other models that we could implement to see if the results of our findings change.

There are other models proposed in [1] which consist of several different types of network perturbation. The degree assortative perturbation model is based on the Chung-Lu random graph model. Edges in this model are selected for addition/deletion with a probability that is proportional to the degrees of the endpoints. That is, if the endpoints have high degree then it is more likely that the edge between them would be deleted or added if there is no edge between them already. The model is biased toward edges that have high degree endpoints, which is the opposite from the uniform perturbation model that we used. Another model presented in [1] is the link prediction based model where the results of a missing link prediction algorithm are used
to identify the edges to add or remove. We have also discussed quite a few of the other models in the
literature review section of this dissertation. It would be very interesting to see how our results for the
uniform perturbation model compare to the same analysis using the various different noise models.

7.4 Other Applications

We also hope to apply our network analysis expertise to problems in cybersecurity. Network vulnerabil-
ity is a central issue for cybersecurity, thus, knowing how susceptible a network is to noise and how easy
it is to destroy it will shed light on to how to keep the network secure against different forms of attack.
Understanding how random/targeted noise/attacks on the network change the parameters of the network,
will make it clearer how to defend the network and make sure it remains operational. Given the high re-
liance we have on computer systems, and the increasing number of attempts to compromise them, high
interest into cybersecurity from the government has resulted in increased availability of funds for research
in cybersecurity (see “NSF awards 74.5 million USD to support interdisciplinary cybersecurity research”
-https://www.nsf.gov/news/newsumm.jsp?cntnId=136481). Possible topics of study include: con-
structing secure networks that are highly resistant to random noise and targeted attacks, developing novel
and extremely efficient search strategies for networks to identify critical components, and formulating formal
methods for classifying how resilient/susceptible a network is to noise based on the network structure.
8 Conclusion

We have presented a novel way of identifying high centrality vertices using group testing. We demonstrated the effectiveness of the method for identifying high betweenness centrality vertices as well as high closeness centrality vertices. In the betweenness centrality case our group testing method is particularly effective in networks that have stable BC ranking under small edge perturbations. This work serves as a preliminary study to demonstrate the potential of group testing in network algorithms.

We have seen that in some networks, the middle ranking vertices can combine to give false positive results. However, this phenomena can be utilized to identify alternate groups of important vertices. For example, if the goal is to disrupt the network and eliminating the highest centrality vertex is disadvantageous, we can use group testing to find alternate groups of vertices that are not as highly ranked, but together can be effective in disrupting the network.

Each test in group testing methods requires computing the centrality value of only the supervertex. For closeness centrality it can be easily done using the standard algorithm by simply performing one BFS starting at the supervertex, and this is the reason for the major speedup results for closeness presented earlier. However, for betweenness centrality this is not the case. Currently, our implementation for betweenness centrality follows the more expensive method of using the Brandes algorithm to compute the BC for all vertices in the compressed network, and then use the value of the supervertex for group testing. Therefore to cut down on the amount of computation we have a proposed an algorithm (which we are still developing) for finding the BC of only a specified vertex (or supervertex) without having to compute the other values. Another approach could be to use an approximation algorithm such as the one described in [2] to estimate the BC of the supervertex only.

It should be noted that our approach would also work for directed and weighted networks. The grouping of vertices remains exactly the same and only the centrality calculation for each test is different. In the worst case we can use the Brandes algorithm version which is for directed and weighted networks though a better approach would involve using a BC algorithm that can compute the BC value of a specific vertex such as the one we have been developing.

Finally, we have also presented results on network perturbation using the uniform perturbation model based on Erdős-Rényi random graphs taken from [1].

While there has been research on how perturbations to the network affect the analysis results, these studies have often resulted in conflicting conclusions. Some of these conflicting opinions are on whether noise equally affects all centrality metrics, whether the change is monotonic or not and what properties of the network affect the stability.
From our results it is evident that betweenness centrality rankings are affected significantly greater than the closeness centrality rankings. The degree centrality measure is the most stable out of the three, and we have provided a theoretical basis for this hypothesis. Moreover, networks for which group testing worked well were stable in their BC rankings under increasing values of perturbation. Conversely, those networks for which group testing was not successful had BC rankings which were easily disrupted by even small levels of noise. This grouping of networks was not evident for closeness centrality.

We performed a more fine-grained analysis by looking at the stability over different sets of top-$k$ ranked centrality vertices. We see that depending on how the experiments are setup many of these conflicting opinions can be reconciled. For example, while noise affects all centrality metrics, for some such as degree the effect is easier to compute, whereas the stability of closeness and betweenness centrality also depend on the network structure. Also, the overall dominant stability value across consecutive $k$s decreases monotonically with increase in noise level, however for individual $k$ this change is non-monotonic. Finally the global properties affect the stability, only if the networks are very different in structure, such as scale-free versus random. However, if the networks fall in same group, e.g. scale-free, then the local connections of high centrality vertices is the chief determinant of stability.

Our experiments on network stability under noise demonstrate two extremely important findings which have never been observed so far. The first is that networks where the high centrality vertices are very well-connected, i.e., they form a “rich-club”, are more stable. The second is that the stability of the rankings of nodes depends on the number of top ranked nodes ($k$) being investigated. The top nodes seem to arrange themselves into groups; if the value of $k$ is such that it does not split a group then the results are stable, otherwise they are unstable.

We believe that our results will have long ranging applications to network analysis. Based on the conditions of stability proposed, users can evaluate the stability of their networks, without applying the noise model and tuned to the number of vertices to be considered and the centrality metric. Users can further use the conditions to improve the stability of their data collection methods.
References


Appendices

A Other Group Testing Designs

Prior to working on group testing for the identification of important vertices in networks, I worked on new superimposed constructions for group testing, as well as multiple access channels. Presented in this appendix are some of the designs and results.

A.1 Construction of a Superimposed Code Using Partitions

The following is a construction of superimposed codes based on a partial order on partitions. The work was originally presented in [72].

Recall the definitions given in section 3.1.1 for code, superimposed code, boolean sum, weight, and intersection. The construction involves all of these definitions, as well as the following regarding partitions.

A.1.1 Partial Order on Set Partitions

Let $n, q_1, q_2 \in \mathbb{N}$ where $2 \leq q_1 \leq q_2 \leq n$. $A_{q_i} = \{0, 1, \ldots, q_i - 1\}$ is the standard $q_i$-ary alphabet, $i \in \{1, 2\}$, and $[n] = \{1, 2, \ldots, n\}$ is a set of $n$ elements. $M_{q_i} = \{\mu_1, \mu_2, \ldots, \mu_{q_i!}\}$ is the set of all $q_i!$ permutations of $q_i$ symbols. Let $x = (x_1, x_2, \ldots, x_n) \in A_{q_i}$ denote an arbitrary $q_i$-ary $n$-sequence that identifies an unordered $q_i$-partition of $n$. A $q_i$-partition is of the form:

$$E_0; E_1; \ldots; E_{q_i}$$

where $E_m = \{i : x_i = m\}$ for some vector $m \in A_{q_i}$ e.g. if $[n] = \{1, 2, 3, 4, 5\}$ and $q_i = 3$ then $x = (1, 1, 1, 2, 0)$ identifies the partition: $\{(5); (1, 2, 3); (4)\}$ where $E_0 = \{5\}, E_1 = \{1, 2, 3\}$ and $E_2 = \{4\}$

Remark. Any $q_i$-partition contains $q_i', 1 \leq q_i' \leq q_i$, nonempty clusters.

For any $\mu \in M_{q_i}$ we can identify a $q_i$-ary $n$-sequence:

$$x^{\mu} = (\mu_1(x_1), \mu_2(x_2), \ldots, \mu_{q_i}(x_n))$$
called a $\mu$-complement of $x$. Notice that all $\mu$-complements of any $x$ identify the same unordered $q_i$-partition. In our construction we want to isolate all partitions of a set that have $q_1$ and $q_2$ nonempty clusters. Then we will define a partial order relation between them. Let us denote a partition with $q_i$ nonempty clusters by $\overline{x}_{q_i}$. The set of all partitions of $[n]$ that contain $q_i$ nonempty clusters will be denoted by $S_{q_i}(n)$. In addition to $\mu$-complement where a bijection acts on a vector, we also would like to introduce this operation by $\phi$ to $x_\phi = (\phi(x_1), \phi(x_2), \ldots, \phi(x_n))$. Let us introduce the following relation $\succ$. We will say that $\overline{x}_{q_1} \succ \overline{x}_{q_2}$ iff $\exists \phi : A_{q_2} \rightarrow A_{q_1}$ a surjection, such that $x_{q_1}^\phi = x_{q_2}$ for some vector $x_{q_2}$ corresponding to $\overline{x}_{q_2}$, and $x_{q_1}$ corresponding to $\overline{x}_{q_1}$. $\overline{x}_{q_2}$ is then called a sub-partition of $\overline{x}_{q_1}$. Notice that if $q_1 < q_2$, then we can never have $\overline{z}_{q_2} \succ \overline{x}_{q_1}$ since we cannot define a surjection from any set to a larger set.

$\phi$ is a function that maps elements from $A_{q_2}$ to $A_{q_1}$. For each surjection $\phi$, there exists an element $x_{q_1}^\phi$ in $A_{q_1}$ such that $x_{q_1}^\phi = x_{q_2}$.
The task is now to find the number of partitions that have \( q_1 \) clusters (nonempty parts) and also the number of partitions that have \( q_2 \) clusters. This is a problem easily solved by using Stirling set numbers of the second kind. We have that the number of partitions of \([n]\) having \( q_1 \) clusters is:

\[
|S_{q_1}(n)| = \frac{1}{q_1!} \sum_{i=0}^{q_1-1} (-1)^i \binom{q_1}{i} (q_1 - i)^n
\]

and likewise:

\[
|S_{q_2}(n)| = \frac{1}{q_2!} \sum_{i=0}^{q_2-1} (-1)^i \binom{q_2}{i} (q_2 - i)^n
\]

### A.1.2 Superimposed Code Construction

The construction of the superimposed code is similar to the construction using the set-subset partial order. Instead of using the set-subset relation, we shall use the partition-subpartition relation as defined above. Let us denote elements from \( S_{q_2}(n) \) by \( P_i \), \( 1 \leq i \leq |S_{q_2}(n)| \), and elements from \( S_{q_1}(n) \) by \( R_j \), \( 1 \leq j \leq |S_{q_1}(n)| \). Consider the following matrix:

\[
\begin{bmatrix}
R_1 & P_1 & \cdots & P_i & \cdots & P_t \\
R_1 & x_{11} & x_{12} & \cdots & x_{1t} \\
\vdots & x_{21} & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & x_{ji} & \ddots \\
R_j & \vdots & \ddots & x_{ji} & \ddots & \vdots \\
R_N & x_{N1} & x_{N2} & \cdots & x_{Nt}
\end{bmatrix}
\]

Where \( x_{ji} = 1 \) iff \( R_j \succ P_i \) else \( x_{ji} = 0 \). Notice that for this code we have:

\[
\begin{align*}
\text{t} &= |S_{q_2}(n)| \\
\text{N} &= |S_{q_1}(n)|
\end{align*}
\]

We can also find the weight and maximum intersection and hence a value for \( s \) – the strength of the code. First of all notice that to find the weight of any column \( i \) we need to find the number of surjections from \( P_i \) to partitions containing \( q_1 \) clusters. This is translated to being the number of surjections from \( A_{q_2} \) to \( A_{q_1} \) .
This value depends only on the values $q_1$ and $q_2$ which means as we increase $n$ hence increase the number of codewords $t$ and their length $N$, we will always have a constant weight (assuming we keep $q_1$ and $q_2$ constant). The number of surjections from a set of size $q_2$ to a set of size $q_1$ is a well known formula:

$$
\sum_{i=0}^{q_1-1} (-1)^i \binom{q_1}{i} (q_1 - i)^{q_2}
$$

Now we have to notice that for any surjection, a permutation composed with the surjection is also a surjection (a different one) but the partition to which it is mapping it’s subject to is the same because the vector is simply a $\mu$-complement. More formally:

$$
x^\phi \approx \tilde{z}_{q_1}
$$

$$
(x^\phi)^{\mu_i} \approx \tilde{z}_{q_1} \forall \mu_i \in M_{q_1}
$$

Also: $\forall \mu_i \in M_{q_1}, \mu_i \neq (1), \mu_i \circ \phi = \phi \neq \phi$ and since the composition of an onto mapping with an onto mapping is again onto, we have that $\phi$ is onto. The weight is going to be the number of surjections from a given partition $P_j$ to distinct partitions. Since the vectors representing partitions of the form $R_j$ contain all $q_1$ symbols, we must divide the sum of surjections by $q_1!$ to obtain $w$:

$$
w = \frac{1}{q_1!} \sum_{i=0}^{q_1-1} (-1)^i \binom{q_1}{i} (q_1 - i)^{q_2} = |S_{q_1}(q_2)|
$$

Notice that the value for $w$ is a Stirling set number i.e. it is the number of ways of partitioning a set of $q_2$ elements into $q_1$ clusters, which is exactly what we are doing in the partial order relation.

To find $\lambda$, the maximum intersection, we must look at partitions $P_i$ and $P_j$ and we must have the partition distance between them to be minimal, i.e. 1. When we have a distance of 1, this means that there exist vectors corresponding to the partitions such that there is one common position in the vectors where the symbols are different. So we can take these two symbols and treat them as one i.e. instead of having $q_2$ symbols we will now have $q_2 - 1$ symbols that will represent the two partitions. Then we ask: how many partitions containing $q_1$ clusters will the partition containing $q_2 - 1$ clusters be mapped to? We have already answered this, it is simply again a Stirling set number:

$$
\lambda = \frac{1}{q_1!} \sum_{i=0}^{q_1-1} (-1)^i \binom{q_1}{i} (q_1 - i)^{q_2-1} = |S_{q_1}(q_2 - 1)|
$$

Now we can calculate $s$ using the Kautz-Singleton bound:

$$
s = \left\lfloor \frac{w - 1}{\lambda} \right\rfloor = \left\lfloor \frac{|S_{q_1}(q_2)| - 1}{|S_{q_1}(q_2 - 1)|} \right\rfloor
$$

Below is a summary of the parameters for the code that we constructed.
\[ t = |S_{q_2}(n)| \quad \quad w = |S_{q_1}(q_2)| \]
\[ N = |S_{q_1}(n)| \quad \quad \lambda = |S_{q_1}(q_2 - 1)| \]

### A.2 Superimposed Codes Analysis for Multiple Access Channels and Group Testing

A multiple access channel is a medium that allows multiple terminal stations to communicate with a central station. We will consider a system containing a large amount of terminal stations and a multiple access OR channel connecting the terminal stations to the central station, as in the slotted ALOHA system [21]. We implement the symmetric model of group testing and superimposed code constructions [38] to the discussed multiple access information transmission model, for the calculation of bounds on the multiple access channel capacity. Recent results in Superimposed Coding Theory provide the possibility of improving the known bounds [18] [40] on the multiple access OR channel capacity as well as the minimum number tests requires in two-stage group testing.

#### A.2.1 Multiple Access Channel

The system studied contains \( M \) terminal stations and a multiple access channel (MAC) connecting the terminal stations to the central station (CS). The MAC is a common medium over which the terminal stations communicate with the CS. Through the study of the limitations on the rate of communication i.e. information transmission rate, that arise from the interference among signals sent to the CS from the terminal stations, we obtain bounds on the rates of information transmission (also known as throughputs) that are achievable for a specific MAC system. The specific system which we are concerned with is known as the slotted ALOHA system [21]. The system, which was implemented at the University of Hawaii, was a ground radio system that allowed users to transmit requests to the CS at any time they desired [40]. By employing recent results in the theory of Superimposed Codes, we will demonstrate a higher lower bound for the maximum throughput (channel capacity) for the multiple access OR channel (MAC-OR) used in the slotted ALOHA system.

#### A.2.2 ALOHA System

Suppose that a system contains \( M \) terminal stations that are connected by a MAC-OR to a CS. Each terminal station has a source, which can generate one of \( t \) binary sequences, that we refer to as information packets. The information packets are of length \( N \) and are transmitted via the MAC-OR as requests to the CS. The CS receives the information sent by the terminal stations, as one information packet, which is the Boolean sum of the sent information packets. It does not distinguish between which station sent the packet since it is only interested in the contents of the information packet. The CS decomposes the received information packet
into the packets that were sent, after which the CS sends out answers (to the requests) as binary sequences of length $K$ via the feedback broadcast channel (FBC) to all $M$ stations simultaneously as shown in the illustration.

In the slotted ALOHA system, the terminal stations are synchronized and transmit information packets in windows. A window is a slot of time whose duration is equal to the exact amount of time required to transmit a single information packet, thus all windows are of equal length. Since the stations are synchronized, information packet transmission begins at the start of a window for all stations wishing to transmit [21] [40]. In this scenario, interference among the information packets sent is restricted to the windows, and thus we must concentrate on how the transmitted information packets interact and the nature of the information packet received by the CS.

**A.2.3 Mathematical Modeling of the MAC-OR and Slotted ALOHA System**

Let $K$ be an integer that corresponds to the minimum number of bits required to encode all possible requests, that the terminal stations may wish to send, using binary sequences. The total number of possible information packets (requests) is $t = 2^K$. Enumerate the $2^K$ information packets using the integers $1, 2, ..., t$. Let $N \in \mathbb{N}$ such that $N \geq K$. Consider the following $N \times t$ binary matrix:

$$X = \begin{bmatrix}
    x_1(1) & x_1(2) & \ldots & x_1(t) \\
    x_2(1) & x_2(2) & \ldots & x_2(t) \\
    \vdots & \vdots & \ddots & \vdots \\
    \vdots & \vdots & \ddots & x_i(j) \\
    \vdots & \vdots & \ddots & \vdots \\
    x_N(1) & x_N(2) & \ldots & x_N(t)
\end{bmatrix} \quad x_i(j) \in \{0, 1\}$$
The above matrix is known as a code of length $N$ and size $t$. Let $x(u) = (x_1(u), x_2(u), ..., x_N(u))$, $u = 1, 2, ..., t$, $x_i(u) \in \{0, 1\}$, for $i = 1, 2, ..., N$, denote the $u^{th}$ column of $X$. $x(u)$ is known as a code packet, which corresponds to the information packet with the number $u$. Note that each information packet has length $N$ and thus we can divide the time window required to transmit one information packet, into $N$ slots of equal length. Each slot corresponds to the amount of time required to transmit one symbol of a binary sequence of length $N$.

A.2.4 MAC-OR Channel Model

Suppose that $m$ stations transmit information packets $x(u_1), x(u_2), ..., x(u_m)$ in a window. The output of the MAC-OR (which is the information packet received by the CS) is described by the following function: $f$:

$$f(x(u_1), x(u_2), ..., x(u_m)) = \begin{bmatrix} f(x_1(u_1), x_1(u_2), ..., x_1(u_m)) \\ f(x_2(u_1), x_2(u_2), ..., x_2(u_m)) \\ \vdots \\ f(x_N(u_1), x_N(u_2), ..., x_N(u_m)) \end{bmatrix}$$ (1)

$$f(x_i(u_1), x_i(u_2), ..., x_i(u_m)) = 1, \quad \sum_{j=1}^{m} x_i(u_j) \neq 0, \quad \sum_{j=1}^{m} x_i(u_j) = 0, \quad i = 1, 2, ..., N.$$ (2)

Note that the function outputs the Boolean sum (component-wise sum using the OR operation) of the input information packets. The output of the MAC-OR which is received by the CS then has to be decomposed (by the CS) into the constituent information packets. An effective method for encoding the information packets into code packets is crucial for the CS to be able to decompose the received packet.

A.2.5 Slotted Aloha System Model

For every window in the slotted ALOHA system, each terminal station performs exactly one of the following actions:
• The station will be silent.

• The station will send one of the \( t \) possible code packets, transmitting one binary symbol per slot.

Let \( p \) be a real number such that \( 0 < p < 1 \). Assuming that the sources of the terminal stations operate independently, for each time window the source of the \( m \)-th station, \( m = 1, 2, \ldots, M \) has two possibilities:

• With probability \( p \), the source generates one information packet which will be sent by the \( m \)-th station in the given time window, using the corresponding code packet.

• With probability \( 1 - p \), the source does not generate anything, and the \( m \)-th station stays silent during the time window.

Let \( \mu > 0 \) be a fixed parameter and let \( \xi_{\mu} \) denote the random variable that corresponds to the number of code packets transmitted during a given window. Since there are \( M \) terminal stations in total, the probability that within any given time window, there will be \( n \) code packets transmitted to the CS is:

\[
\Pr\{\xi_{\mu} = n\} = \binom{M}{n} p^n (1 - p)^{M - n}, n = 0, 1, 2, \ldots M
\]  

(3)

\( \mu \) is selected such that \( Mp = \mu \) and as \( M \to \infty \) and \( p \to 0 \), \( \mu \) remains fixed. Note that as \( M \to \infty \), (3) becomes the Poisson distribution with parameter \( \mu \) (average number of requests per time window) and thus for \( M \) sufficiently large, we have:

\[
\Pr\{\xi_{\mu} = n\} = \frac{\mu^n e^{-\mu}}{n!}, n = 0, 1, 2, \ldots
\]  

(4)

The CS has threshold (also known as the capacity of the CS) \( T \). \( T \) is the maximum number of requests that the CS can answer i.e. it is the maximum number of information packets that can be sent out by the CS within one window. If in any given time window, \( N \) symbols can received by the CS then the number of answers (information packets of length \( K \)) which can be sent out i.e. the threshold is:

\[
T = \frac{N}{K}
\]  

(5)

Note that if too many information packets are transmitted through the MAC-OR to the CS within one time window, the CS will not be able to answer any of them i.e. refusal occurs. The method for encoding the information packets (to be transmitted through the MAC-OR) which allows the CS to decompose the received packet into the constituents, arises from Group Testing [16] and Superimposed Coding Theory [38], [22].
A.2.6 MDS Codes

Let \( F_q = \{0, 1, \ldots, q - 1\} \) be a set of \( q \) distinct elements. \( F_q \) is known as the alphabet and is often taken to be \( \mathbb{Z}_q \) (integers by \( \text{mod} \, p \)) when \( q \) is a prime number. \( F_q^n \) is the set consisting of vectors of length \( n \) that are built from the elements of the alphabet \( F_q \).

The Hamming distance between two vectors \( x, y \in F_q^n \) is the number of places in which they differ and is denoted by \( d(x, y) \).

A \( q \)-nary code \( C \), of length \( n \), size \( t \), and minimum distance \( d \), is a subset of \( F_q^n \) such that \( d = \min\{d(x, y) | x, y \in F_q^n, x \neq y\} \). \( C \) is referred to as an \((n, t, d)\) code.

Assume \( F_q \) is the Galois field \( GF(q) \), the set of elements endowed with two operations (addition + and multiplication \( \cdot \)), containing 0 and 1, and where any equation of the form \( a \times x + b = c \), \( a, b, c \in GF(q) \), has a solution. Then \( q \) has to be a prime power and we regard \( F_q^n \) as the vector space \( V(n, q) \).

A linear code over \( GF(q) \) is a subspace of \( V(n, q) \), for some positive integer \( n \). If \( C \) is a \( k \)-dimensional subspace of \( V(n, q) \), of minimum distance \( d \), then \( C \) is referred to as an \([n, k, d]\) code.

A \( k \times n \) matrix \( G \) whose rows form a basis of a linear \([n, k, d]\) code is called a generator matrix of the code.

A parity-check matrix \( H \) for an \([n, k, d]\) code \( C \) is an \((n - k) \times n\) matrix satisfying \( GH^T = 0 \), where \( H^T \) denotes the transpose of \( H \) and \( 0 \) is an all zero \( k \times (n - k) \) matrix. Then we have that: \( C = \{x \in V(n, q) | xH^T = 0\} \).

It should be noted that for any linear \([n, k, d]\) code \( C \), \( d \leq n - k + 1 \). This arises from the fact that \( C \) is a \( k \)-dimensional space of size \( q^k \) and from the Singleton bound [51], it follows that \( q^k \leq q^{n-d+1} \). Also, the minimum weight of the code matches the minimum distance i.e. \( d = w \).

**MDS Code:** A linear code with minimum distance \( d = n - k + 1 \) is an \([n, k, n - k + 1]\) code known as a maximum distance separable code or MDS code for short.

An additional property of MDS codes is that they are separable i.e. the number of digits in a codeword can be separated into \( k \) independent information digits and \( n - k \) dependent check digits. This means that the intersection between any two codewords can be at most \( k - 1 \) since two codewords that have an intersection of \( k \) or higher, would be equal (since we can assign the information bits to be the ones in which the two codewords are equal).

**Reed-Solomon Code:** Let \( k, n \) be integers that satisfy \( 1 \leq k < n \leq q + 1 \), and let \( a_1, a_2, \ldots, a_{q-1} \) denote the non-zero elements of \( GF(q) \). The matrix:
is the parity-check matrix of an MDS \([n, k, n - k + 1]\) code which is known as the Reed-Solomon code.

Using \([n, k, n - k + 1]\) Reed-Solomon codes where \(n\) is maximal i.e. \(n = q + 1\), the family of superimposed codes implemented in the slotted ALOHA system is obtained.

### A.2.7 Superimposed Codes from Reed-Solomon Codes

Let \(q\) be a prime or prime power and \(k \geq 2\). Take \(C\) to be a \(q\)-nary \([q + 1, k, q - k + 2]\) Reed-Solomon code, of size \(t = q^k\). Code \(C\) can be represented by the following matrix, whose columns are the codewords of \(C\):

\[
C = \begin{bmatrix}
    y_1(1) & y_1(2) & \cdots & y_1(t) \\
    y_2(1) & y_2(2) & \cdots & y_2(t) \\
    y_3(1) & y_3(2) & \cdots & y_3(t) \\
    \vdots & \vdots & \ddots & \vdots \\
    y_{q+1}(1) & y_{q+1}(2) & \cdots & y_{q+1}(t)
\end{bmatrix}
\]

\(y_i(j) \in GF(q), i = 1, 2, \ldots, q + 1, j = 1, 2, \ldots, q^k\)

\(C\) can be transformed into a binary superimposed code by applying the following transformation [38]. Each symbol of \(GF(q)\) is associated with a binary column vector of length \(q\) and weight 1 i.e.

\[
\begin{bmatrix} 0 & 1 & 2 & 3 & \cdots & q-1 \end{bmatrix} = \begin{bmatrix}
    1 & 0 & 0 & 0 & \cdots & 0 \\
    0 & 1 & 0 & 0 & \cdots & 0 \\
    0 & 0 & 1 & 0 & \cdots & 0 \\
    0 & 0 & 0 & 1 & \cdots & 0 \\
    \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}
\]
Each symbol in \( C \) is then replaced by the binary column associated with it. This transformation produces a binary \( q(q+1) \times q^k \) matrix \( X \), which is a superimposed code of size \( t = 2^K \) with the following parameters (strength is calculated by (5) using \( w = q + 1, \lambda = k - 1 \)):

\[
K = \lfloor k \log_2 q \rfloor, \quad N = q(q+1), \quad s = \left\lfloor \frac{q}{k-1} \right\rfloor
\]  

(7)

This binary superimposed code will be labeled as an \((N, s, t)\) code.

A.2.8 Slotted ALOHA System Performance

Assume that \( \xi = p \) code packets are transmitted in a given window. Denote the packets by \( x(u_1), x(u_2), \ldots, x(u_p) \), where \( 1 \leq u_1 \leq u_2 \leq \ldots \leq u_p \leq t \). Let \( z = f(x(u_1), x(u_2), \ldots, x(u_p)) \) be the packet received by the CS. Recall that \( z \) is the symmetrical Boolean sum of the transmitted code packets.

Let \( 1 \leq s \leq T \leq t = 2^K \) be integers. Recall that the CS has a threshold of \( T \) packets (maximum number of requests which can be answered during one window). The CS uses a binary superimposed \((N, s, t)\) code \( X \) for the decomposition of \( z \) into the constituent code packets. The system operates in the following manner. Within a given window, \( p \) terminal stations transmit the information packets (as code packets \( x(u_1), x(u_2), \ldots, x(u_p) \) ) via the MAC-OR to the CS. The CS receives their Boolean sum \( z \) and selects all \( p' \) columns of \( X \) which are covered by \( z \). Two situations arise:

- If \( p' \leq T \) then the CS transmits (via the FBC) answers (of length \( K \) bits) to the requests covered by \( z \).
  (Successful transmission).

- If \( p' \geq T + 1 \) then the CS does not answer any of the requests received within the window. (Refusal).

Note that if \( p \leq s \) then \( p' = p \) and \( z \) will cover only the \( p \) code packets which were transmitted, but when \( p > s \) then \( p' \geq p \) and other code packets (not in the \( p \)-set transmitted) could be covered by \( z \). By studying the average number of code packets covered by an arbitrary \( p \)-set of the code \( X \), we will improve the bounds [40], [18] on the channel capacity of the MAC-OR. Before establishing the new result, we introduce some of the definitions and notations needed and the previous results.

A.2.9 Characteristics of the System

The maximal possible answers that the CS can transmit over the FBC is \( T \). This means that the CS answers (if any) not more than \( T - p \) unnecessary requests (assuming \( \xi = p \) code packets were transmitted). For a refusal to occur, we must have (by the definition of a superimposed code) \( \xi \geq s + 1 \) and thus a lower bound on the expected value of successfully transmitted requests within a window of length \( N \) is [21]:
\[ A_s(\mu) = \sum_{p=0}^{s} \frac{p\mu^p e^{-\mu}}{p!} \]  

(8)

Where \( \mu \) is the average number of requests per time window. For the specific superimposed code \( X \) used by the CS, denote by \( E(\mu, X) \) the average number of successfully transmitted information bits per slot of the time window. Since each request contains exactly \( K \) information bits, then using (7), it is evident that [21]:

\[ E(\mu, X) \geq \frac{K}{N} A_s(\mu) = \frac{K}{N} \sum_{p=0}^{s} \frac{p\mu^p e^{-\mu}}{p!}, \]

(9)

\( E(\mu, X) \) is the rate of the superimposed \((N, s, t)\) code \( X \). Note that by optimizing \( A_s(\mu) \) by \( \mu \) (with restrictions imposed on \( \mu \)), and then optimizing the bound on \( E(\mu, X) \) by the parameters used for \( X \), the lower bound on the channel capacity of the MAC-OR i.e. the maximal rate of information transfer per slot of the time window, is obtained. The optimization introduced by us requires results obtained in [50] which we shall examine in a later section. First, we take note of the bound obtained for the channel capacity of the MAC-OR in [40] and [18].

A.2.10 Previous Lower Bounds on Channel Capacity

Let \( C_T \) denote the channel capacity of the MAC-OR. When \( T = 1 \), then the system we study becomes the slotted ALOHA system without feedback i.e. \( N = K \) and the superimposed code is not used. Under these circumstances, Kleinrock [40] showed that:

\[ C_1 \geq e^{-1} \approx 0.368, \]

(10)

When \( T \geq 2 \), the Reed-Solomon superimposed is implemented and \( N > K \). Using the random coding bound obtained by D’yachkov-Rykov-Antonov, D’yachkov-Rykov went on to show that [18]:

\[ \lim_{T \to \infty} C_T \geq \frac{\log_2 e}{e} \approx 0.5307, \]

(11)

In the next section we establish the results obtained in [50], which will be used to improve the lower bound on the MAC-OR channel capacity.

A.2.11 Improving the Lower Bound on Channel Capacity

We can improve the lower bounds established in [40], [18], by increasing the sum: \( \frac{K}{N} A_s(\mu) \). That is, instead of summing to \( s \), the strength of the code, we can assume that a higher quantity of code packets can be received
and answered. By taking into account the threshold of the system \( T \), and finding the average number of extra codewords covered by an arbitrary \( p \)-set of \( X \), we can use Markov’s inequality to estimate the probability that an arbitrary \( p \)-set will be answered and hence increase the average number of successfully transmitted requests in a time window and thus increase the lower bound on channel capacity.

**Average Number of Code Packets of \( X \), Covered by an Arbitrary \( p \)-set [50]**

Consider an arbitrary MDS code \( C \) with parameters \( q,k,n \) of volume \( t = q^k \), \( k \leq n-1 \leq q \) and codewords \( x(i) = \{ x_1(i), x_2(i), ..., x_n(i) \}, i = 1,T \). Where \( q \) is the number of symbols in the alphabet \( GF(q) \), \( n \) is the length of the code, \( d = n-k+1 \) is the minimal Hamming distance and the minimal weight of the codewords. Denote by \( S_w(n) \) the number of codewords in \( C \) of weight \( w \). Then from [51] we have:

\[
S_w(n) = \binom{n}{w} (q-1) \sum_{j=0}^{w-d} (-1)^j \binom{w-1}{j} q^{w-d-j}, w = d,n \tag{12}
\]

Let \( \mathbf{0} = (0, 0, ..., 0) \) be the zero codeword in \( C \) and let \( x(i) \ast x(j) = \{ x_1(i) \cdot x_1(j), x_2(i) \cdot x_2(j), ..., x_n(i) \cdot x_n(j) \} \) be the component wise dot-product of codewords \( x(i) \) and \( x(j) \) (is the multiplication operation used in \( GF(q) \)). The set of codewords \( \{ x(u_1), x(u_2), ..., x(u_p) \} \) does not cover \( \mathbf{0} \) if \( x(u_1) \ast x(u_2) \ast ... \ast x(u_p) \neq \mathbf{0} \). Let \( C_0(p,n) \) denote the number of possible \( p \)-sets from \( C \) which do not cover \( \mathbf{0} \). Let \( \binom{q^k}{p} = r \) and take \( \{ p_1, p_2, ..., p_r \} \) to be the set of all possible \( p \)-sets of \( C \). Let \( p(C) = \{ x(i_1), x(i_2), ..., x(i_{k_1}) \}, i = 1,r \) denote the set of codewords of \( C \) that are not in \( p_i(C) \) but are covered by it. Then the average number of codewords that **do not belong to but are covered** by an arbitrary \( p \)-set of \( C \) is:

\[
\frac{\sum_{j=0}^{k_1} k_j}{\binom{q^k}{p}} \tag{13}
\]

By changing the order of summation we obtain the result of [50]. Instead of summing over the sizes of the \( p \)-sets we can sum over the frequencies of appearance in the \( p \)-sets of the codewords \( x(u_1), x(u_2), ..., x(u_{q^k}) \). Note by the symmetrical property of the MDS code \( C_0(p,n) = C_{x(u_1)}(p,n) = C_{x(u_2)}(p,n) = ... = C_{x(u_{q^k})}(p,n) \). Then for each of the \( q^k \) vectors of \( C \), the number of \( p \)-sets which cover it has to be \( \binom{q^k-1}{p} - C_0(p,n) \), thus allowing us to write (13) in the form presented in [50]:

\[
L(p) = \left( \frac{\binom{q^k-1}{p} - C_0(p,n)}{\binom{q^k}{p}} \right) q^k \tag{14}
\]
Note that \( L(p) \) is the same in \( X \), the binary superimposed code built from \( C \). In order to calculate \( L(p) \) we must be able to calculate \( C_0(p,n) \). Denote by \( D(p,v) \) number of \( p \)-sets in \( C \) for which the arbitrary but fixed \( v \) symbols are all non-zero, then [50]:

\[
D(p,v) = \left\{ \begin{array}{ll}
\left(\frac{q^{k-v}(q-1)^v}{p}\right) & \text{if } v \leq k \\
\binom{S_v(v)}{p} & \text{if } v > k
\end{array} \right.
\]

where \( S_v(v) \) is given by (11) and is the number of codewords with weight \( v \) in an MDS \((q,k,v)\)-code.

Recall that an MDS \([q,k,n]\)-code is separable and so any \( k \) positions of this code can be regarded as information positions and thus, for \( v \leq k \), the number of vectors which have nonzero entrances on all \( v \) fixed information positions is \( q^{k-v}(q-1)^v \). Then the number of \( p \)-sets in \( C \) with the property required for \( D(p,v) \) is the number of ways of choosing \( p \) elements from \( q^{k-v}(q-1)^v \) (hence the first piece of \( D(p,v) \)). For \( v > k \) we fix \( v \) arbitrary strings in the \([q,k,n]\) MDS code and consider a new code \( Y \) which has the same volume but whose codewords are the \( v \) strings taken from \( C \). Then number of codewords with nonzero entrances on all \( v \) fixed positions is equal to \( S_v(v) \) — numbers codewords with weight \( v \) in \( Y \) (hence the second piece of \( D(p,v) \)).

Using the principle of inclusion and exclusion, we can now calculate the total number of \( p \)-sets of \( C \) that have at least one non-zero element in their dot-product i.e. the total number of \( p \)-sets of \( C \) that do not cover \( \emptyset \) or \( C_0(p,n) \) [50]:

\[
C_0(p,n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} D(p,i)
\]

A.2.12 Lower Bound on Channel Capacity of MAC-OR in the Slotted ALOHA System

Let \( \varsigma_p \) be the random variable that corresponds to the number of extra codewords of the superimposed code \( X \) (specified by (7)) that are covered by a \( p \)-set. Then by using Markov’s inequality, we have the following:

\[
\Pr\{\varsigma_p \geq T + 1 - p\} \leq \frac{L(p)}{T + 1 - p}
\]

Where \( T \) is the threshold of the system. Note that when \( \varsigma_p \geq T + 1 - p \), the number of requests that the CS selects becomes greater than or equal to \( T + 1 \) and refusal occurs. For our derivation we take:

\[
\Pr\{\varsigma_p \geq T + 1 - p\} = \frac{L(p)}{T + 1 - p}
\]

Then:

\[
\Pr\{\varsigma_p \leq T - p\} = 1 - \frac{L(p)}{T + 1 - p}
\]
Note that when $\Pr\{\varsigma_p \leq T - p\} = 0$, then the system will always refuse since the threshold will be exceeded each time. Hence for each set of code parameters $\{q,K\}$, we wish to find $p_{\text{max}}$ such that $\Pr\{\varsigma_{p_{\text{max}}} \leq T - p_{\text{max}}\} > 0$ or:

$$L(p_{\text{max}}) < T + 1 - p_{\text{max}}$$

(20)

Then by combining (19),(20), and (8), we obtain that the expected value of successfully transmitted requests within a window of length $N$ is:

$$A_{p_{\text{max}}} (\mu) = \sum_{p=0}^{p_{\text{max}}} \frac{p\mu^p e^{-\mu}}{p!} \Pr\{\varsigma_p \leq T - p\} = \sum_{p=0}^{p_{\text{max}}} \frac{p\mu^p e^{-\mu}}{p!} \left( 1 - \frac{L(p)}{T + 1 - p} \right)$$

(21)

Where $\mu$ is the average number of requests per time window. For the specific superimposed code $X$, $E(\mu, X)$ the average number of successfully transmitted information bits per slot of the time window becomes:

$$E(\mu, X) \geq \frac{K}{N} A_{p_{\text{max}}} (\mu) = \frac{K}{N} \sum_{p=0}^{p_{\text{max}}} \frac{p\mu^p e^{-\mu}}{p!} \left( 1 - \frac{L(p)}{T + 1 - p} \right),$$

(22)

The lower bound on the rate of the $(N,s,t)$ superimposed code used for the slotted ALOHA system then becomes:

$$\mathcal{C}(q,K) \geq \max_{\mu} \left\{ \frac{K}{N} A_{\text{p_{max}}} (\mu) \right\} = \frac{K}{N} A_{p_{\text{max}}} (\mu_{\text{max}})$$

(23)

Denote by $\mathcal{C}^*(q,K)$, the series as in (23) but without the term arising from the Markov inequality (19). $\mathcal{C}^*(q,K)$ was used initially in calculations to estimate the possible rate for the given code. When the estimated rate significantly improved the bound (11), bound (23) was calculated.

The lower bound on the channel capacity of the MAC-OR for the slotted ALOHA system becomes:

$$\mathcal{C}(q) \geq \max_{K,\mu} \left\{ \frac{K}{N} A_{p_{\text{max}}} (\mu) \right\} = \frac{K_{\text{max}}}{N} A_{p_{\text{max}}} (\mu_{\text{max}})$$

(24)

**Theorem.** Let $p = \alpha q$, $k = \left\lfloor \frac{q}{\log_2 q} \right\rfloor$. Then as $q \to \infty$:

$$\lim_{q \to \infty} L(\alpha q) \to \begin{cases} \infty & \text{if } \alpha > \ln 2 \\ 0 & \text{if } \alpha < \ln 2 \end{cases}$$

(25)

**Proof:** Take:
Let \( q \to \infty \), and \( p = \alpha q \) where \( \alpha \) is a constant. As \( q \to \infty \), we have that:

\[
L(p) = \frac{\left(q^k - 1\right)}{p^k} - \sum_{i=1}^{q+1} (-1)^{i+1} \binom{q+1}{i} D(p, i) q^k
\]

and so we have that \( D(p, i) = \left(q^k - (q-1)\right) \). Since \( k = \left\lfloor \frac{q}{\log_2 q} \right\rfloor \), then \( q^k = q^{\log_2 2^k} \approx 2^q \) as \( q \to \infty \). Take into account that \( q^k - i(q-1)^i = q^k(1 - \frac{1}{q})^i \), then as \( q \to \infty \):

\[
D(p, i) \approx \frac{q^k(1 - \frac{1}{q})^i}{p^i} = \frac{q^k(1 - \frac{1}{q})^i}{p^i} \approx \frac{2^q e^{-\alpha i}}{p^i}
\]

Also, take into account that \( \left(q^k - 1\right)^i \left(q-1\right)^i = q^k(1 - \frac{1}{q})^i \), then as \( q \to \infty \):

\[
\lim_{q \to \infty} L(\alpha q) = \lim_{q \to \infty} \frac{2^q 2^q a q \sum_{i=0}^{q+1} (-1)^i \binom{q+1}{i} e^{-\alpha i}}{2^q a q} = 2^q(1 - \frac{1}{e^\alpha})^q + \begin{cases} 
\infty & \text{if } \alpha > \ln 2 \\
0 & \text{if } \alpha < \ln 2 
\end{cases}
\]

thus the Theorem is proved. We have since produced a more rigorous proof of this theorem and it is still to be published.

**Corollary.** Let \( \mu = q \ln 2 \), \( k = \left\lfloor \frac{q}{\log_2 q} \right\rfloor \). Then:

\[
\mathcal{C} \geq \lim_{q \to \infty} \mathcal{C}(q) \geq \ln 2 \approx 0.69315
\]  \hspace{1cm} (26)

**Proof:**

Recall from (7) that \( N = q(q+1) \) and since \( k = \left\lfloor \frac{q}{\log_2 q} \right\rfloor \), then \( K = [k \log_2 q] = q \) and so:

\[
\lim_{q \to \infty} \mathcal{C}(q) \geq \lim_{q \to \infty} \frac{q}{q(q+1)} A_{\max}(\mu)
\]

From our Theorem we know that \( L(\alpha q) \to 0 \) when \( q \to \infty \) and \( \alpha < \ln 2 \) and thus we can conclude that:

\[
\lim_{q \to \infty} A_{\max}(q \ln 2) = q \ln 2 \implies q \ln 2
\]

implying that:

\[
\lim_{q \to \infty} \mathcal{C}(q) \geq \lim_{q \to \infty} \frac{q}{q(q+1)} A_{\max}(\mu) = \lim_{q \to \infty} \ln 2 \frac{q^2}{q(q+1)} = \ln 2
\]
A.2.13 Application to Group Testing

The obtained bound on the channel capacity of the MAC-OR allows us to calculate the asymptotic upper bound on the number of tests required in two-stage group testing. Let \( t \) be the size of the population, and let \( p \) denote the number of defective units among the population. \( N \) is the number of tests required to find the defective units. If \( p = \ln 2 \log_2 t \) then as \( t \to \), the following holds:

\[
N \leq \frac{p}{\ln^2 t} \log_2 t = (\log_2 t)^2
\]

This is a significant improvement of the previous upper bound on the number of tests required.

Similarly, if \( N = (\log_2 t)^2 \), then as \( t \to \), the following holds:

\[
p \leq \ln 2 \log_2 t
\]
Figure 20: **Karate Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)

**B Supplementary Results in Network Noise Analysis**

**B.1 Additional Noise Charts**

The following are more detailed charts showing the effect of more levels of noise (than those presented in section 4) for more $k$-values. The centralities considered are degree, betweenness, and closeness. The networks for which the results are shown are some of those used in the analysis when using multiple values of $k$ in section 4. The main purpose of these charts is to support the results already presented at a more fine-grained level.
Figure 21: **Chesapeake Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)

Figure 22: **LesMis Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)
Figure 23: **Celegans Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)

Figure 24: **AS1 Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)
Figure 25: **AS2 Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)

Figure 26: **GrQc Network Perturbations** - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)
Figure 27: HepTh Network Perturbations - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)

Figure 28: Power Grid Network Perturbations - Betweenness (BC), Closeness (CC), Degree, Noise model: xER (addition and deletion), aER (addition), dER (deletion)