Mining and analysis of real-world graphs

Armita Abedijaberi

MINING AND ANALYSIS OF REAL-WORLD GRAPHS

by

ARMITA ABEDIJABERI

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2018

Approved by:

Dr. Jennifer Leopold, Advisor
Dr. Dan Lin
Dr. Wei Jiang
Dr. Simone Silvestri
Dr. Egemen Cetinkaya

PUBLICATION DISSERTATION OPTION

This dissertation consists of the following five articles, formatted in the style used by the Missouri University of Science and Technology:

Paper I: Pages 6 to 24,“FSMS: a Frequent Subgraph mining algorithm using Mapping Sets”, was published in 12th International Conference on Machine Learning and Data Mining MLDM, Springer, Cham, pg. 761-773, 2016.

Paper II: Pages 25 to 55, “All-inclusive Frequent Free Subtree Mining using Automorphism List”, is in submission to International Conference on Information and Knowledge Management (CIKM), 2018.

Paper III: Pages 56 to 76, “Interactive Visualization of Robustness Enhancement in Scale-free Networks with Limited Edge Addition (RENEA)”, was published in Proceedings of the 23rd International Conference on Distributed Multimedia Systems, pg. 34-42, 2017.

Paper IV: Pages 77 to 103, “Interactive Visualization of Robustness Enhancement in Scale-free Networks against Cascading Failures”, is under revision, Journal of Visual Languages and Computing (JVLC) special issue on Best Papers from DMSVLSS2017, Elsevier, 2018.

Paper V: Pages 104 to 123,“Motif-level Robustness Analysis of Power Grids”, is in preparation for IEEE ICDM International Workshop on Data Mining in Networks (DaMNet), 2018.

ABSTRACT

Networked systems are everywhere - such as the Internet, social networks, biological networks, transportation networks, power grid networks, etc. They can be very large yet enormously complex. They can contain a lot of information, either open and transparent or under the cover and coded. Such real-world systems can be modeled using graphs and be mined and analyzed through the lens of network analysis. Network analysis can be applied in recognition of frequent patterns among the connected components in a large graph, such as social networks, where visual analysis is almost impossible. Frequent patterns illuminate statistically important subgraphs that are usually small enough to analyze visually. Graph mining has different practical applications in fraud detection, outliers detection, chemical molecules, etc., based on the necessity of extracting and understanding the information yielded. Network analysis can also be used to quantitatively evaluate and improve the resilience of infrastructure networks such as the Internet or power grids. Infrastructure networks directly affect the quality of people's lives. However, a disastrous incident in these networks may lead to a cascading breakdown of the whole network and serious economic consequences. In essence, network analysis can help us gain actionable insights and make better data-driven decisions based on the networks. On that note, the objective of this dissertation is to improve upon existing tools for more accurate mining and analysis of real-world networks.

ACKNOWLEDGMENTS

Completion of this Ph.D. dissertation was possible with the support of several people. Firstly, I would like to express my sincere gratitude to my advisor Dr. Jennifer Leopold, for the continuous support throughout the course of my doctoral research. I would also like to thank my committee members, Dr. Dan Lin, Dr. Wei Jiang, Dr. Simone Silvestri, and Dr. Egemen Cetinkaya, for their insightful comments and encouragement.

Finally, I would like to thank my family who have patiently supported me throughout my life and helped me out with anything that I have ever needed.

TABLE OF CONTENTS

Page

PUBLICATION DISSERTATION OPTION iii

ABSTRACT iv

ACKNOWLEDGMENTS v

LIST OF ILLUSTRATIONS xi

LIST OF TABLES xvi

SECTION

1. INTRODUCTION 1

1.1. GRAPHS AND NETWORKS 1

1.2. NETWORK ROBUSTNESS 4

1.3. FREQUENT SUBGRAPH MINING 5

1.4. SUMMARY 5

PAPER

I. FSMS: A FREQUENT SUBGRAPH MINING ALGORITHM USING MAPPING SETS 6

1. INTRODUCTION 7

2. RELATED WORK 9

3. FSMS 11

3.1. ALGORITHM 16

3.2. ALGORITHM COMPLEXITY 17

4. EXPERIMENTS 19

4.1. RESULTS 19

5. FUTURE WORK 20

6. CONCLUSION 22

REFERENCES 22

II. ALL-INCLUSIVE FREQUENT FREE SUBTREE MINING USING AUTOMORPHISM LIST 25

1. INTRODUCTION 26

2. PRELIMINARIES 27

3. BACKGROUND AND RELATED WORK 30

4. METHODOLOGY 32

4.1. SUBTREE REPRESENTATION 32

4.2. EXTENSION LABEL 34

4.3. AUTOMORPHISM LISTS 35

4.4. SMAL ALGORITHM 36

4.5. HOW TO AVOID FALSE NEGATIVES 39

4.5.1. Removing the Infrequent Vertices/Edges 40

4.5.2. Removing the Examined Frequent Edges 42

4.6. HOW TO AVOID DUPLICATE CANDIDATES 42

4.6.1. Expansion from Left to Right 42

4.6.2. Keeping the Examined Frequent Edges 44

5. DISTRIBUTED COMPUTATION IN SPARK 46

6. EXPERIMENTAL EVALUATION 48

6.1. RUNTIMES DON'T SAY EVERYTHING 50

7. CONCLUSIONS AND FUTURE WORK 53

REFERENCES 53

III. INTERACTIVE VISUALIZATION OF ROBUSTNESS ENHANCEMENT IN SCALE-FREE NETWORKS WITH LIMITED EDGE ADDITION (RENEA) 56

1. INTRODUCTION 56

2. RELATED WORK 58

2.1. RECONFIGURATION WITH EDGE REWIRING 59

2.2. RECONFIGURATION WITH EDGE ADDITION 61

2.3. NETWORK VISUALIZATION 62

3. ROBUSTNESS ENHANCEMENT ALGORITHM 63

3.1. ROBUSTNESS METRIC 63

3.2. RENEA 64

3.3. EDGE REMOVAL 67

3.4. THE GRAPHICAL USER INTERFACE 68

4. EXPERIMENTS 69

4.1. RESULTS 70

5. A RUNNING EXAMPLE IN RENEA 71

6. CONCLUSION 74

REFERENCES 75

IV. INTERACTIVE VISUALIZATION OF ROBUSTNESS ENHANCEMENT IN SCALE-FREE NETWORKS AGAINST CASCADING FAILURES 77

1. INTRODUCTION 77

2. RELATED WORK 79

2.1. RECONFIGURATION WITH EDGE REWIRING 80

2.2. RECONFIGURATION WITH EDGE ADDITION 82

2.3. NETWORK VISUALIZATION 84

3. MODELING THE CASCADING FAILURES IN SCALE-FREE NETWORKS 84

4. ROBUSTNESS ENHANCEMENT ALGORITHM 87

4.1. ROBUSTNESS METRIC 87

4.2. EFFICIENCY 88

4.3. RENEA 88

4.4. EDGE WEIGHT ASSIGNMENT 92

4.5. THE GRAPHICAL USER INTERFACE 94

5. EXPERIMENTS 96

5.1. RESULTS 97

6. CONCLUSION 99

REFERENCES 101

V. MOTIF-LEVEL ROBUSTNESS ANALYSIS OF POWER GRIDS 104

1. INTRODUCTION 104

2. PRELIMINARIES 105

2.1. NETWORK MOTIFS 106

2.2. ASSESSING STATISTICAL SIGNIFICANCE OF MOTIFS 106

2.3. CENTRALITY METRICS 108

2.4. MALICIOUS ATTACKS 109

2.5. TOPOLOGICAL ROBUSTNESS METRICS 110

3. EUROPEAN POWER GRIDS 110

3.1. EUROPEAN POWER GRIDS ROBUSTNESS 111

4. EXPERIMENTS 113

5. CONCLUSION 119

REFERENCES 120

SECTION

2. FUTURE WORK 124

2.1. GRAPH MINING USING DISTRIBUTED PLATFORMS 124

2.2. MOTIF-BASED ANALYSIS OF GRAPH ROBUSTNESS 125

3. CONCLUSION 125

BIBLIOGRAPHY 127

VITA 129

LIST OF ILLUSTRATIONS

Figure Page

PAPER I

1. Example of a graph where a particular vertex (i.e., g) will only be included in a frequent subgraph when minimum support is ≥ 2 if we allow isomorphic subgraphs to share edges (i.e., (g,m)). 8

2. Example of a graph G with 13 vertices and 18 edges, where each vertex (edge) has a *label* and a unique *id*. Solid lines outline all the instances of one-edge subgraph $\{R\text{-}a\text{-}B\}$ in G, where R and B are the vertex labels and a is the edge label. A dashed line outlines one of the instances of subgraph $\{\{$R-a-B$\}, \{$B-b-Y$\}, \{$Y-a-R$\}, \{$Y-c-C$\}\}$ in G. 13

3. Example of *bijection* mapping among the vertices of 3 isomorphic one-edge subgraph $\{$R-a-B$\}$ in graph G in [Figure 2]. 14

4. In this example only a part of a bigger graph is shown. Assume that after iteration 1 in FSMS, two subgraphs $\{A$-a-B$\}$ and $\{$B-s-C$\}$ have been detected as frequent. Each of them have their own *Key*, *Values* and *MappingSets*. During the iteration 2, using completely different *MappingSets*, two instances have become similar. To avoid counting an instance more than once, FSMS uses a hash table. 16

5. Comparing the performance of FSMS vs. FSG for frequent subgraph mining in graphs with different numbers of labelled vertices and labelled edges. For both algorithms the same minimum support was selected. However we had to keep the minimum support threshold high to be able to compare the performance of FSG and FSMS; this was because FSG was not able to finish the processing for low minimum support. 20

PAPER II

1. Example of an induced subtree $T1$ and an embedded subtree $T2$ of graph G. Each vertex is shown with its label (an uppercase alphabetic character) and its unique id (a positive integer). An induced subtree preserves the parent-child relation between two vertices. An embedded subtree preserves the ancestor relations but not necessarily the parent relations for vertices in G. 29

2. Example of a graph where all vertices and edges are labeled and each vertex has a unique id as well. When minimum support is ≥ 2, a particular vertex (i.e., v_1) will only be included in a frequent subtree if we allow isomorphic subtrees to share edges (i.e., $(e_{1,3})$). 30

3. Example of a 1-subtree ($T1$) and a 2-subtree ($T2$) in graph G. Each subtree is represented with an automorphism list that contains lists of instances (in terms of vertex ids) for that subtree in graph G. 34

4. Example of extension labels for a vertex in a subtree. Subtree T can be extended by three different extension labels through v_2..................... 35

5. Example of a subtree T that has an automorphism (i.e., T is isomorphic to itself). T has two instances in G where each instance has two sets of automorphism. Hence, the automorphism list of T contains 4 lists. 37

6. A running example of SMAL... 40

7. An example to show that keeping the infrequent edges results in detecting larger frequent subtrees that would have been missed otherwise. In this example, if infrequent edges get removed from G, then $T2$ and $T3$ will not be detected as frequent subtrees... 41

8. Examples of valid ✓and invalid $\times$ connections 45

9. An example to show that keeping the instances of frequent edges even after they are processed results in detecting larger frequent subtrees, which would have been missed otherwise.. 47

10. Workflow of Spark Implementation of SMAL 49

11. Results for Citeseer Graph.. 50

12. Results for Citation Graph.. 51

13. Comparing the number of frequent subtrees................................. 52

PAPER III

1. RENEA GUI 68

2. RENEA GUI after uploading the US Air graph, all the specifications of this graph are presented in the GUI. 69

3. x-axis represents the number of removed nodes and y-axis represents the size of LLC in the American Airline network before and after reconfiguration. Line in red shows the performance of RENEA...................... 71

4. x-axis represents degrees of nodes and y-axis represents how many nodes share the same degree. After adding 50 edges to US Air network, its degree distribution remains almost the same................................ 72

5. A graph with 20 nodes and 21 edges is uploaded........................... 72

6. A list of suggested edges to add in order to enhance its robustness. 73

7. Graph after adding 3 edges. 73

8. 3 edges are nominated to get removed. 74

9. After adding 3 edges and removing 3 edges, the overall robustness improvement is 25%. 74

PAPER IV

1. The standard deviation of betweenness vs. the Weight Parameter (Beta) for a scale-free network with 1000 vertices and 1567 edges with degree exponent of 2.5. Increasing the value of β from -1 to $+1$ results in decreasing the value of δ_B. 95

2. RENEA GUI 96

3. In this figure, x-axis represents the number of edges that are added to a network and y-axis represents the size of Largest Connected Component (LCC) after removal of hubs. The performance of RENEA against four different algorithms is presented in this plot. In all networks, RENEA outperforms the other four algorithms, i.e. networks' elements stay more connected in case of attacks on hubs. 98

4. The network robustness as a function of Alpha and Beta. In this figure, x-axis represents the tolerance parameter (Alpha), y-axis represents the weight parameter (Beta), and the red colors show the area of high robustness. 99

PAPER V

1. Eight motifs on 3 and 4 nodes, respectively. Numbers in parentheses are the Motif Identification numbers [29]. 107

2. Motif concentration for different motifs in different networks is presented in this plot. The x-axis represents the value of γ which increases as the fragility of networks increases. 114

3. The percentage of connectivity loss vs. the percentage of node removal in different types of node attacks. 115

4. The 5-node motifs that are only significant in the Germany and Italy power grids. Numbers in parentheses are the Motif Identification numbers [29]. 115

5. Motif concentration decay for significant motifs vs. the node removal percentage in degree-based node attacks. 117

6. Z-score of significant motifs as a function of the percentage of removed edges in random-based vs. betweenness-based edge attack. 119

LIST OF TABLES

Table Page

PAPER I

1. Performance of FSMS vs. FSG algorithm on chemical datasets with different graph properties. Columns in this table represent the number of vertices, the number of edges, the number of distinct vertex labels, the number of distinct edge labels, minimum support, total number of discovered frequent subgraphs of different sizes, the maximum size of subgraphs found by FSMS and FSG, and running time of FSMS and FSG in seconds, respectively. A dash in the table means we had to abort the computation for FSG because it was taking too long to complete execution. 21

PAPER II

1. Datasets and their characteristics 49

PAPER III

1. The performance of RENEA vs. three other algorithms Algo1, Algo2, and Algo3. Algo1 adds edges randomly, Algo2 adds edges among low-degree nodes, and Algo3 adds edges among high-degree nodes. The data set is American Airline dataset with 332 nodes, 2126 edges and initial robustness of 0.1079. We added 20, 30, 40, 50, and 60 edges to the original graph and the averages of total robustness improvement acquired after running each algorithm are presented. 70

PAPER IV

1. The performance of RENEA on different networks with N vertices and L edges. $< k >$ represents the average degree in a network. R and *Eff* represent the obtained robustness and efficiency after applying RENEA and edge-weight assignment. m indicates the maximum percentage of added edges. We added less than 2 percent of the number of edges in the original graphs to each network. 100

PAPER V

1. The global characteristics of two robust and two fragile European power grids are presented. These power grids are ordered by increasing γ (the exponential degree distribution characteristic parameter). N, M, and $\langle k \rangle$

represent the number of nodes, the number of edges, and the mean nodal degree, respectively. 112

2. The z-score as a qualitative measure of statistical significance of each motif for each network is presented in this table. NS denotes that the motif is not significant. 113

SECTION

1. INTRODUCTION

We live in a connected world. In our social and digital lives, we encounter networks (or graphs) on a daily basis. Online social networks such as Facebook are based on gigantic networks in which people are connected through so-called friendship links. Browsing the Internet means traversing a large network of pages that is connected via clickable (hyper) links. Accessing one webpage on a mobile phone creates a few dozen wired or wireless connections between devices in a matter of micro seconds. Other examples include networks of publications linked by citations, transportation networks, metabolic networks, and communication networks. Networks are everywhere around us, and influence the way in which we communicate, socialize, search, navigate and consume information. Graphs are mathematical structures used to represent/model pairwise relations between objects in these real world networks. When networks are stored in a digital format, they can produce an enormous amount of data. Such a large volume of data is sometimes called big data, not only because of its quantity, but also because the data may arrive at an enormous speed and because the data are usually diverse in terms of what type of information is represented. Data are used in many disciplines of science to verify hypotheses about a certain domain. Within the field of computer science, we specifically consider tasks related to storing, retrieving, manipulating and understanding data in an automated and efficient way.

1.1. GRAPHS AND NETWORKS

A graph is one of the most fundamental data structures used to describe the relationship between objects within a certain domain. This structure is very simple, consisting of a number of dots that are called vertices in graph terminology and nodes in network terminology. Between some of these dots are lines that are called edges in graph terminology and links in network terminology. Network terminology is generally used in situations where you want to think of transporting/sending things along the links between nodes, whether those things are physical objects (e.g. road networks) or information (e.g. computer networks). Graph terminology is more often used in situations where you want the edges to represent other types of relationships between the vertices. A well-known example of a real-world graph is a social network, in which a vertex represents a person, and an edge represents a social relationship between two people that it connects.

There are a number of very different ways in which researchers analyze graphs/networks. One approach, which is more closely associated with the network side of things than the graph side, focuses on understanding individual nodes and how they are related to the entire network. For example, there is the question of centrality [3]: which node would we expect to have the most traffic flowing past it, whether that means an overloaded server on a computer network, or the most important person in a social network. Another example is the question of robustness of a network [4]: how robust a network is with respect to removing nodes with the highest centrality. These approaches are primarily concerned with the global topology of networks and their reliability. A second approach, which is more closely associated with the graph side of things than the network side, focuses on thinking of a graph/network as a single geometric object and extracting the structural information of interest by inspecting the way vertices are connected together. For example, frequent subgraph mining [5] consists of discovering interesting patterns in graphs. These approaches are primarily concerned with understanding the local structure of graphs. A third approach, which lies in between the network side and the graph side of things, is concerned with the analysis of local network structure and the effect on the global topology of a network to unveil hidden mechanisms behind vulnerability of complex networks.

There are different types of graphs including regular, trees [9], random [8], small-world [7], and scale-free [6] networks. Two extreme kinds of graphs are regular lattices (meshes) and trees. These are usually man-made networks that have the lowest heterogeneity (e.g. the number of connections each node has is more or less the same) and the lowest randomness (the probability of any two randomly selected nodes to be connected to each other is very low or zero). Regular graphs tend to have long average paths and high clustering (as the nodes tend to be densely connected in groups).

Another extreme type of graphs is called random ER (Erdos-Renyi) graphs, which are generated by starting with a disconnected set of nodes that are then paired with a uniform probability. Such random networks will have low heterogeneity (most nodes have the same number of connections), and the degree distribution will be a Gaussian bell-shaped curve. Random graphs built after the ER algorithm have short average paths and low clustering [10]. Most real-world networks, however, especially social networks, do not have a homogeneous degree distribution that is mostly found in regular or random networks. The number of connections each node has in most networks varies greatly and they are positioned somewhere between regular and random networks. In fact, Watts and Strogatz (1998) [7] proposed a model where the connections between the nodes in a regular graph were rewired with a certain probability. The resulting graphs were between regular and random in their structure and are referred to as small-world (SW) networks. SW networks are structurally very close to many social networks in that they have a higher clustering and almost the same average path than the random networks with the same number of nodes and edges. SW networks usually have high modularity (groups of the nodes that are more densely connected together than to the rest of the network).

Finally, there is a large class of so-called scale-free (SF) networks characterized by a highly heterogeneous degree distribution, which follows a *power-law* (Barabasi & Albert 1999) [6]. They are called scale-free, because zooming in on any part of the distribution does not change its shape; there are a few, but significant number of nodes with a lot of connections and there is a trailing tail of nodes with a very few connections at each level of magnification.

Such networks have been suggested as representative models of complex systems in areas ranging from the social sciences (e.g., collaboration graphs of movie actors or scientific coauthors) to molecular biology (e.g., cellular metabolism and genetic regulatory networks) to the Internet (e.g., web graphs and router-level graphs).

These structures combine heterogeneity and randomness; they can have low or high modularity [12], and many SW networks are also scale-free.

1.2. NETWORK ROBUSTNESS

Scale-free networks are everywhere. They can be seen in airline traffic routes, connections between actors in Hollywood, weblog links, personal relationships, and terrorist networks. A scale-free network obeys a power-law distribution [11] in the number of connections between nodes on the network. Some nodes exhibit extremely high connectivity while the vast majority are relatively poorly connected.

The proliferation of scale-free networks and our increasing dependence on them (particularly given their prevalence in energy, transportation, and communications systems) begs the question: how reliable are these networks?

Scale-free networks are extremely tolerant of random failures [15]. In a random network, a small number of random failures can collapse the network. However, a scale-free network can absorb random failures of up to 80% of its nodes before it collapses. The reason for this is the inhomogeneity of the nodes on the network. So, failures are much more likely to occur on relatively small nodes.

On the other hand, scale-free networks are extremely vulnerable to intentional attacks on their hubs [14]. Attacks that simultaneously eliminate as few as 5% of a scale-free network's hubs can collapse the network.

Considering error tolerance and attack vulnerability, which are two common and important properties of scale-free networks, extensive research efforts have been made to study the robustness of such networks, which is defined as the ability of surviving nodes and edge to remain interconnected, and design algorithms to enhance the overall robustness of such networks.

1.3. FREQUENT SUBGRAPH MINING

Graph mining is critical to a variety of application domains with the primary goal of extracting statistically significant [13] and useful knowledge from data. Its basic objective is to discover the hidden and useful data patterns from a very large set of data. Graph mining finds its applications in various problem domains, including: bioinformatics, chemical reactions, program flow structures, computer networks, social networks etc. The research goals in frequent subgraph mining are directed at: i) effective mechanisms for generating candidate subgraphs (without generating duplicates) and ii) how best to process the generated candidate subgraphs so as to identify the desired frequent subgraphs in a way that is computationally efficient and procedurally effective.

1.4. SUMMARY

Graphs are everywhere. From computers on the Internet to infected patients in an epidemic to molecular processes within a cell, complex webs of interactions between many different entities play critical roles in both society and the natural world. The objective of this dissertation is to mine and analyze the structure of real-life networks. In this dissertation we explore five research problems in this broad area. Papers I and II focus on mining frequent subgraphs in a large graph. Papers III and IV focus on the robustness of complex networks. Paper V studies on how local subgraphs contribute to the overall robustness of complex networks.

PAPER

I. FSMS: A FREQUENT SUBGRAPH MINING ALGORITHM USING MAPPING SETS

Armita Abedijaberi and Jennifer Leopold

Department of Computer Science

Missouri University of Science and Technology, Rolla, Missouri 65401

With the increasing prevalence of data that model relationships between various entities, the use of a graph-based representation for real-world problems offers a logical strategy for organizing information and making knowledge-based decisions. In particular, often it is useful to identify the most frequent patterns or relationships amongst the data in a graph, which requires finding frequent subgraphs. Algorithms for addressing that problem have been proposed for over 15 years. In the worst case, all subgraphs in the graph must be examined, which is exponential in complexity, and subgraph isomorphisms must be computed, which is an *NP-complete* problem.

Frequent subgraph algorithms may attempt to improve the actual runtime performance by reducing the size of the search space, avoiding duplicate comparisons, and/or minimizing the amount of memory required for compiling intermediate results. Herein we present a frequent subgraph mining algorithm that leverages mapping sets in order to eliminate the isomorphism computation during the search for non-edge-disjoint frequent subgraphs. Experimental results show that absence of isomorphism computation leads to much faster frequent subgraph detection when there is a need to identify all occurrences of those subgraphs.

1. INTRODUCTION

Data are used every day to make knowledge-based decisions for the purposes of marketing, emergency management, law enforcement, and other applications. Large companies, such as Google, use data to provide predictive, quick searches based on related data [1]. Governments use a variety of inter-related data to help predict terrorist attacks [2]. Data mining also has been used in the health sciences to correlate multiple gene expression patterns for cancer to identify patients who are most at risk [3]. As more data become available, relationship mapping has become even more integral. The use of a graph-based representation for real-world problems offers a logical strategy for organizing information and making knowledge-based decisions. Currently, graph data are used for visualizing computer networks [4], biological networks [5], and the Internet [6].

One of the most common applications of graph data mining (GDM) is to identify the most frequent relationships or patterns amongst the data in a graph, which requires finding frequent subgraphs. In a *graph-transaction setting*, the input will be a collection of relatively small graphs, which necessitates that the search for frequent subgraphs be performed over each individual graph in the collection before those results can be combined; this is in contrast to a *single-graph setting* where the input is a single (presumably large) graph. Formally, we define the Frequent Subgraph Mining (FSM) problem using Definitions 1-4 given below.

Definition 1. A labelled graph $G = (V, E, L_V, L_E)$ consists of a set of vertices V, a set of undirected or directed edges E, and two labeling functions L_V and L_E that associate labels with vertices and edges, respectively. It should be noted that the labels of any two vertices (or any two edges) may not be unique. However, each vertex (and each edge) will have a unique id.

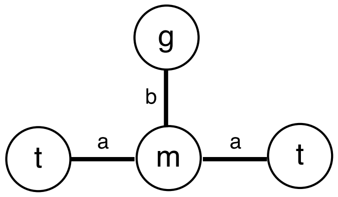


Figure 1: Example of a graph where a particular vertex (i.e., g) will only be included in a frequent subgraph when minimum support is ≥ 2 if we allow isomorphic subgraphs to share edges (i.e., (g,m)).

Definition 2. A graph S = (V_S, E_S, L_{V_S}, L_{E_S}) is a subgraph of $G = (V, E, L_V, L_E)$ iff $V_S \subseteq V$, $E_S \subseteq E$, $L_{V_S}(v) = L_V(v)$ and $L_{E_S}(e) = L_E(e)$ for all $v \in V_S$ and $e \in E_S$.

Definition 3. A subgraph isomorphism of S to G is a one-to-one function f: $V_S \longrightarrow$ V where $L_{V_S}(v) = L_V(f(v))$ for all vertices v $\in V_S$, and for all edges $(u,v) \in E_S$, $(f(u), f(v))$ $\in E$ and $L_{E_S}(u,v) = L_E(f(u), f(v))$.

Definition 4. Let I_S be the set of isomorphisms of a subgraph S in graph G. Given a minimum support threshold τ, the frequent subgraph mining problem (FSM) is to find all non-edge-disjoint subgraphs S in G such that $|I_S| \geq \tau$.

The advantage of limiting frequent subgraphs to only those with disjoint edges is computational tractability [7]. But this comes at the expense of disregarding potentially useful information. Consider the example shown in [Figure 1]. If only edge-disjoint isomorphic subgraphs are considered, the vertex labelled g will not be included in any frequent subgraph when minimum support is ≥ 2. However, if we allow isomorphic subgraphs to share edges, then the subgraph containing the vertices labelled t, m, and g will have frequency 2. The organization of this paper is as follows. Section 2 provides a brief overview of related work in graph data mining, focusing on graph theory based approaches. In Section 3, we discuss the FSMS algorithm. The results of running an implementation of our algorithm on a variety of graphs are provided in Section 4. Finally, we discuss our plans for future work in Section 5 and conclusions in Section 6.

2. RELATED WORK

In [8], the authors divided existing GDM algorithms into three main categories: Graph Theory Based, Inductive Logic Programming, and Greedy Search. Our research focuses on the Graph Theory Based category, which consists of two main groups: Apriori-based approaches and pattern growth-based approaches. Algorithms in the first group generate candidate subgraphs by joining two frequent subgraphs of size k (where k is the number of vertices or the number of edges, depending on the definition used in the particular algorithm). Two subgraphs must contain the same size $k-1$ connected subgraphs to form a new candidate subgraph of size $k+1$ in a level-wise manner. Pattern growth algorithms generate candidates by adding a new edge to a smaller frequent subgraph in all possible extensions.

AGM [9], an Apriori-based graph mining algorithm, uses a breadth-first search to discover frequent subgraphs. It starts with frequent subgraphs consisting of one edge, and iteratively combines frequent subgraphs of equal size to form larger frequent subgraphs. The algorithm is based on the fact that a $k+1$ subgraph cannot be frequent if its immediate parent subgraph of size k is not frequent [9]. FSG [10] is another Apriori-based approach that uses breadth-first search to find all frequent subgraphs by incrementally expanding size k frequent subgraphs by one edge to generate $k+1$ sized candidate subgraphs. Both algorithms do not scale well to large graphs because they tend to incur significant computational overhead due to large candidate pools and subgraph isomorphism comparisons either explicitly or implicitly. The breadth-first search strategy has an advantage in that it allows for the pruning of infrequent subgraphs at an early stage in the FSM process. However, that gain in efficiency comes at the cost of high I/O and memory usage.

Yan et al. proposed gSpan in [11], a pattern growth-based approach that uses depth-first search and a lexicographic ordering to map each subgraph pattern to construct a search tree of frequent subgraphs. The search code tree is built in a depth-first manner by one-edge expansion of the parent vertex. This tree only contains frequent subgraphs that have a smaller depth-first search code than any other isomorphic subgraph in the tree, thus reducing search cost. Additionally, gSpan uses a hash table to keep track of subgraphs already searched, thereby circumventing the processing of any subgraph more than once. Using an efficient coding of graphs, finding previously examined subgraphs is fast. Due to the use of a depth-first search approach, gSpan saves on memory usage in exchange for less efficient pruning.

A more recently proposed graph mining framework, GraMi [12], also employs a pattern growth-based approach. It avoids the costly enumeration of candidate generation performed in Apriori-based methods by adopting the depth-first search code tree methodology of gSpan, and evaluates the frequency of a subgraph as a constraint satisfaction problem. In addition, the authors propose a heuristic search combined with optimizations to improve the performance of the algorithm, employing strategies to prune the search space, postpone searches, and explore special graph types. However, GraMi will not necessarily find all frequent subgraphs because it will only extend a subgraph by frequent edges. GraMi can only guarantee that it will not return infrequent subgraphs (i.e., no false positives).

The above mentioned algorithms take different approaches to perform subgraph isomorphism comparison and to reduce the number of candidate subgraphs generated. Due to their computational complexity, these algorithms typically are used in graph-transaction settings where the individual graphs are relatively small in terms of number of vertices. In general, pattern growth methods tend to be faster than Apriori-based methods, with gSpan being one of the fastest of all FSM algorithms [13]. Like gSpan and GraMi, our algorithm, FSMS, is a pattern growth approach. FSMS iteratively extracts the frequent subgraphs using breadth-first search. However, it differs from other FSM algorithms in that it does not compute isomorphisms, which is the most expensive part of frequent subgraph mining. Instead, it keeps track of isomorphic subgraphs by using a $Mapping\breve{}Set$ system. In terms of runtime, FSMS cannot be compared fairly to gSpan, because gSpan cannot tell us how many occurrences of each frequent subgraph exist in a graph nor can the algorithm easily be modified to do so; in contrast, FSMS returns the actual instances of each frequent subgraph found in a graph. Hence, the preference for which algorithm to use (i.e., gSpan or FSMS) depends on the needs of the application. There is also another framework, MRFSE [14], bulit based on gSpan algorithm. MRFSE uses MapReduce to parallelise gSpan applicable on large datasets. MRFSE keeps the embedding for each frequent subgraph and instead of performing isomorphism, it compares their minimum DFS code [11] to check whether they are isomorphic. However FSMS keeps the bijection between vertices in the frequent subgraphs and group the equivalent vertices together using $MappingSets$. Hence, generated candidates in a group are already isomorphic. The current version of FSMS can be applied on relatively big datasets running on a centralized machine very efficiently.

3. FSMS

FSMS can be applied to a directed or undirected graph $G = (V, E, L_V, L_E)$ that consists of a set of vertices V, a set of edges E, and two labeling functions L_V and L_E that associate labels with vertices and edges, respectively. The algorithm is based on an adjacency list representation of the graph. We assume that the label function $L_V(L_E)$ facilitates an efficient lookup of all vertices (edges) with a particular label (since those labels are not unique); similarly, we assume that, given a vertex (edge), we can find its label in $O(1)$ time.

In discussions where distinction between vertex (edge) ids is not important, we will refer to vertices (edges) simply by their labels. In the process of frequent subgraph mining, calculation of the frequency of a subgraph candidate involves subgraph isomorphism which is a *NP-complete* problem and is considered as a bottleneck for the algorithm. To tackle this issue, herein we present a novel idea for frequent subgraph mining without performing any subgraph isomorphism, which consequently leads to a faster process that is applicable on large graphs.

The FSMS algorithm consists of k iterations in order to find all frequent subgraphs of different sizes from 1 to k. During the procedure of subgraph mining, by the end of iteration i, instances of frequent subgraphs of size i are found and candidate subgraphs of size $i+1$ will be built from them.

The novelty of FSMS candidate generation process comes in replacing isomorphism computation by a *Mapping Sets* system to simplify subgraph frequency counting. This idea is based on the fact that when two or more isomorphic subgraphs expand from the equivalent vertices (in terms of $bijection$ between isomorphic subgraphs) with same-labeled vertices and same-labeled edges, the extended graphs will be automatically isomorphic to each other.

In graph theory, an isomorphism of graphs G and H is a *bijection* between the vertex sets of G and H. A *bijection* is a one-to-one correspondence between the elements of two sets, where every element of one set is paired with exactly one element of the other set, and every element of the other set is paired with exactly one element of the first set. There are no unpaired elements. This kind of bijection is also known as *edge-preserving bijection*.

The FSMS algorithm starts with finding the frequent edges in the graph G as frequent subgraphs of size one. Each frequent subgraph S is represented by a triple of $(Key, Values, MappingSets)$. A *Key* is a set of $labels$ of vertices in S, *Values* is a list of instances of S, and a $MappingSet$ is a set with respect to each vertex in S that contains *ids* of corresponding vertices in instances of S in accordance with the *bijection* among them.

For example, in [Figure 2] when $\tau = 2$, edge $\{R\text{-}a\text{-}B\}$ is a frequent subgraph S of size 1 with 3 instances in the graph G including $\{(1,3)\}$, $\{(7,8)\}$ and $\{(10,11)\}$ where each instance is sorted in lexicographic order. This frequent subgraph S can be represented by *Key*, $Values$ and $MappingSets$ as follows:

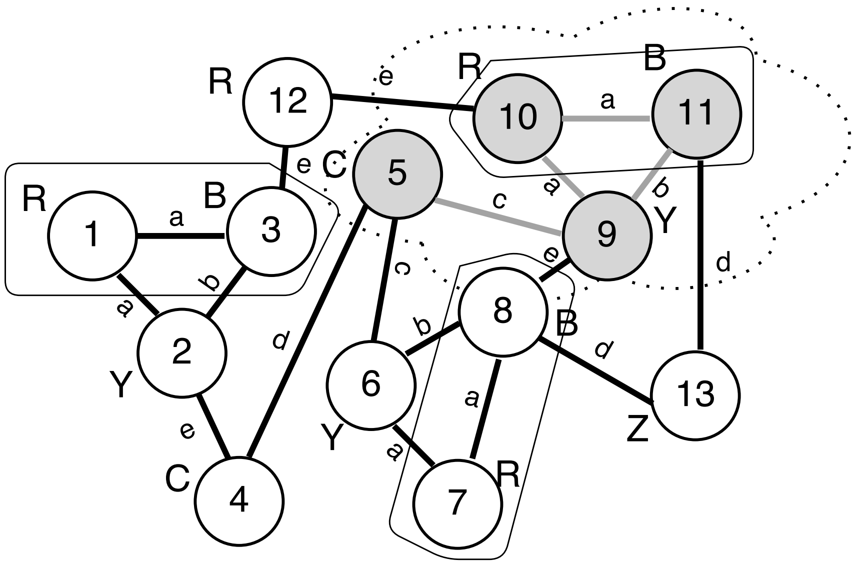


Figure 2: Example of a graph *G* with 13 vertices and 18 edges, where each vertex (edge) has a *label* and a unique *id*. Solid lines outline all the instances of one-edge subgraph $\{R\text{-}a\text{-}B\}$ in *G*, where *R* and *B* are the vertex labels and *a* is the edge label. A dashed line outlines one of the instances of subgraph {{R-a-B},{B-b-Y}, {Y-a-R}, {Y-c-C}} in *G*.

$$Key : \{(R-a-B)\}, \quad Values : \{\{(1,3)\},\{(7,8)\},\{(10,11)\}\}$$

$$MappingSets : \left\{ \begin{array}{l} MS(R) : \{1,7,10\} \\ MS(B) : \{3,8,11\} \end{array} \right\}$$

In one-edge subgraph *S* [Figure 2], for each vertex, a *MappingSet* is associated with it that includes those vertex *ids* in all instances of *S* that correspond to each other, $MS[l] = \{d_1, d_2, ..., d_n\}$ where d_i represents a unique vertex *id*. This means that all of these vertices are equivalent and play the same role in *S*, hence they are grouped together in a *MappingSet* such as $MS(R) : \{1,7,10\}$. In other words, a *bijection* function for 3 instances of *S* maps 3 corresponding vertices (i.e. one vertex per each instance) to each other [Figure 3]. All *MappingSets* in regard to a frequent subgraph have the same length. Consequently, the number of all *MappingSets* and *Values* for a frequent subgraph represents the size and the number of instances for that particular subgraph, respectively.

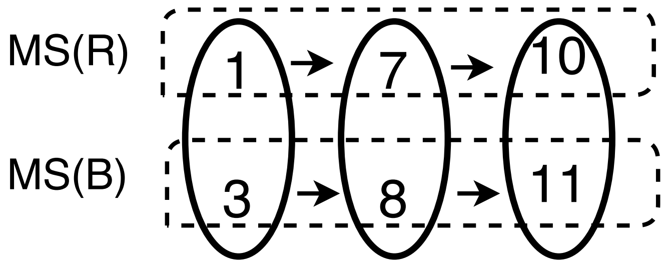


Figure 3: Example of *bijection* mapping among the vertices of 3 isomorphic one-edge subgraph {R-a-B} in graph *G* in [Figure 2].

Candidate subgraphs of size i+1 are generated from subgraph *S* of size i by iterating through all *MappingSets* of *S*. If the size of *Values* for *S* (i.e. frequency of *S* in *G*) is greater than or equal to the frequency threshold τ, *S* is considered frequent and during the next iteration of FSMS it will be used to generate candidate subgraphs of larger size. For each *MappingSets MS* of *S*, new candidate subgraphs are formed as follows: neighbors of vertices in each *MS* with the same label that are connected via same-labeled edges are grouped together. Since all vertices in one *MS* have the same role in *S*, if they extend with links that share the vertex label and the edge label, all the extended subgraphs will be automatically isomorphic to each other.

For example, suppose FSMS is in iteration 4, and a frequent subgraph of size 4 such as {{R-a-B}, {B-b-Y}, {Y-a-R}, {Y-c-C}} is found in *G*, outlined by dashed lines in [Figure 2]. This subgraph has 2 instances in the graph and is represented as:

$$Key : \{\{R-a-B\},\{B-b-Y\},\{Y-a-R\},\{Y-c-C\}\}$$

$$Values : \left\{ \begin{matrix} \{(5,6),(6,7),(6,8),(7,8)\}, \\ \{(5,9),(9,10),(9,11),(10,11)\} \end{matrix} \right\}$$

$$MappingSets : \left\{ \begin{matrix} MS(R) : \{7,10\}, MS(B) : \{8,11\} \\ MS(Y) : \{6,9\}, MS(C) : \{5,5\} \end{matrix} \right\}$$

As depicted in [Figure 2], in $MS[B]$ two of the vertices, 11 and 8 have a connection with vertex label Z and edge label d, and this connection is not already part of the subgraph {{R-a-B}, {B-b-Y}, {Y-a-R}, {Y-c-C}}. In those instances of a subgraph that 8 and 11 belongs to, we add an edge with label d from a vertex with label B to a vertex with label Z; hence 2 instances of a subgraph {{R-a-B}, {B-b-Y}, {Y-a-R}, {Y-c-C}, {B-d-Z}} will be generated which are known to be isomorphic to each other. For the new candidate subgraph, *Key*, *Values* and *MappingSets* are shown below:

$$Key : \{\{R-a-B\},\{B-b-Y\},\{Y-a-R\},\{Y-c-C\},\{B-d-Z\}\}$$

$$Value : \left\{ \begin{matrix} \{(5,6),(6,7),(6,8),(7,8),(8,13)\}, \\ \{(5,9),(9,10),(9,11),(10,11),(11,13)\} \end{matrix} \right\}$$

$$MappingSets : \left\{ \begin{matrix} MS(R) : \{7,10\}, MS(B) : \{8,11\} \\ MS(Y) : \{6,9\}, MS(C) : \{5,5\} \\ MS(Z) : \{13,13\} \end{matrix} \right\}$$

As mentioned, all possible candidate subgraphs of size i+1 are generated after iteration of all available mapping sets of frequent subgraphs of size i. However, it is possible that among the candidate subgraphs, some of the *Values* might become duplicated, due to the fact that instances of different subgraphs might become exactly the same after extension [Figure 4]. To avoid listing the same instance of a candidate multiple times, a hash table is used to keep track of visited subgraphs. A key in this hash table is a sorted list

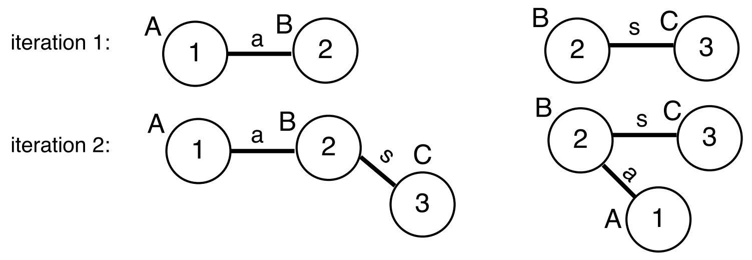


Figure 4: In this example only a part of a bigger graph is shown. Assume that after iteration 1 in FSMS, two subgraphs $\{A$-a-B$\}$ and $\{$B-s-C$\}$ have been detected as frequent. Each of them have their own *Key*, *Values* and *MappingSets*. During the iteration 2, using completely different *MappingSets*, two instances have become similar. To avoid counting an instance more than once, FSMS uses a hash table.

of vertex ids, and values for this key are values of different subgraphs generated with those ids that have been encountered so far. In the hash table, each instance value is represented as a list of tuples sorted in a lexicographic order, where each tuple is representing an edge in the instance by vertex ids at both ends of that edge. Each time the *Values* of a candidate subgraph are generated, FSMS will search for each value in the hash table using the sorted distinct vertex ids in that value. If a generated instance already exists in the hash table, it means that instance has already been encountered, so the value for that instance will be discarded. Otherwise, that value remains and the hash table will be updated. FSMS uses an in-memory pattern growth approach, and to perform frequent subgraph extraction on a centralized machine more efficiently by the end of each iteration the content of the hash table can be deleted.

3.1. ALGORITHM

Input to the FSMS algorithm is a graph $G = (V, E, L_V, L_E)$ and the value of minimum support τ for determining frequent subgraphs. Output from the algorithm is a set of frequent subgraphs, each of which contains at least one edge. In a graph-transaction setting, the FSMS algorithm would be executed for each transaction graph in an input collection, and then the resulting set of frequent subgraphs over the entire collection would be the intersection of the result sets obtained for each individual graph. The FSMS algorithm initially builds a set containing all frequent edges in G based on the value of minimum support. Each frequent subgraph regardless of the size is represented as a triple of $(key, values, mappingSets)$. At this point all frequent subgraphs of size one have been found, and FSMS looks for candidate subgraphs that are one edge larger in size. Starting with a frequent subgraph S, each mappingSet MS contains a list of vertex *ids* where each belongs to one of the instances of S. By grouping neighbors of vertices in MS based on their label and their edge label, FSMS generates candidates of larger size. There also is *hashFreqSGs* which is a hash table that keeps track of subgraphs that have been found. So any time a new instance of a subgraph is found, it is checked through the $hashfreqSubgraph$. If the length of values for each key is greater than or equal to the minimum support, that subgraph will be added to $freqSubgraphs$ and *hashFreqSGs*. Otherwise it will be discarded. This process continues until FSMS finds no more candidates.

3.2. ALGORITHM COMPLEXITY

Let N and n be the number of vertices of graph G and subgraph S, respectively. In general, most algorithms that find frequent subgraphs (e.g., AGM, FSG) are composed of two parts. One part requires, in the worst case, the generation of all subgraphs of G, which takes $O(2^{N^2})$ time. The second part performs the calculation of the frequency of a subgraph S in G, which involves the computation of subgraph isomorphisms. Subgraph isomorphism is NP-complete with $O(N^n)$ time complexity. Therefore, the complexity of frequent subgraph mining is $O(2^{N^2}N^n)$, which is exponential in the problem size. However, one advantage of FSMS over other approaches is that it replaces isomorphism computation by creating and updating Mapping Sets for each frequent subgraph during each iteration. In the worst case, each MappingSets contains n MSs, one MS per vertex. For each MS, FSMS finds neighbors of n vertices and group them together. We are using adjacency list instead of adjacency matrix to store graph G,

Algorithm 1: FSMS

```
1: freqEdges: set of all frequent edges in $G$
2: freqSubgraphs: set of frequent single-edge subgraphs
3: mostRecentSubgraphs $\leftarrow$ freqSubgraphs
4: hashFreqSGs $\leftarrow \varnothing$
5: while mostRecentSubgraphs $! = \varnothing$ do
6:    newSubgraphs $\leftarrow \varnothing$
7:    for subgraph S $\in$ mostRecentSubgraphs do
8:       for mapSet MS $\in$ S.mappingSets do
9:          group neighbors of vertices $\in MS$
10:         for each group g do
11:            candidKey $\leftarrow$ extended S.key
12:            candidValues $\leftarrow$ extended S.values with id $\in g$
13:            for value $\in$ candidValues do
14:               index = a sorted list of distinct vertex ids in value
15:               if index $\notin$ hashFreqSGs then
16:                  hashFreqSGs[index] $\leftarrow \varnothing$
17:               else if candidValues $\in$ hashFreqSGs[index] then
18:                  candidValues = candidValues \value
19:               else
20:                  hashfreqSGs[index] $\leftarrow$ hashFreqSGs[index] $\cup$ value
21:               end if
22:            end for
23:            candidMappingSets $\leftarrow$ extended S.mappingSets
24:            if $|candidValues| \geq \tau$ then
25:               $S'$ = (candidKey, candidValues, candidMappingSets)
26:               newSubgraphs $\leftarrow$ newSubgraphs $\cup S'$
27:            end if
28:         end for
29:      end for
30:   end for
31:   hashFreqSGs $\leftarrow \varnothing$
32:   mostRecentSubgraphs $\leftarrow$ newSubgraphs
33:   freqSubgraphs $\leftarrow$ freqSubgraphs $\cup$ newSubgraphs
34: end while
35: return freqSubgraphs
```

so for each vertex it takes $O(\frac{m}{n})$ to enumerate its neighbors and group them since the neighbor's labels are scanned and aggregated in one pass. Hence, the total complexity for each Mapping Sets in order to be updated for the next iteration is $O(mn)$. There are maximum number of $m-1$ iteration and the average number of discovered frequent subgraphs during each iteration is c. Therefore, the overall complexity of FSMS is cm^2n.

4. EXPERIMENTS

In this section, we experimentally evaluate FSMS, and for performance evaluation comparison, we have also implemented FSG [10]. Both FSMS and FSG follow a pattern growth approach and they return all the occurrences of frequent subgraphs of different sizes. All experiments are conducted using Python 3.3 on a Linux (Ubuntu 14.04) machine with 8 cores running at 2.67GHz with 64GB RAM. For the experiments we used real chemical datasets [15] with different numbers of vertices and edges. There are no multi-edges (two or more edges that are incident to the same two vertices) or self-loops (an edge from a vertex to itself) in the graphs. These datasets contain connected and disconnected graphs with labelled vertices and labelled edges.

4.1. RESULTS

The difference between the amount of time required by FSG and FSMS to find all the frequent subgraphs of graphs with various sizes is shown in [Figure 5]. In all of the experiments the minimum support for both FSG and FSMS was the same and they both discovered the same number of frequent subgraphs. However due to the isomorphism-free nature of FSMS, the performance of these two algorithms is drastically different from each other. In some cases, we aborted the computation for FSG because it was not able to finish the mining process in a reasonable amount of time. In our experiments, several

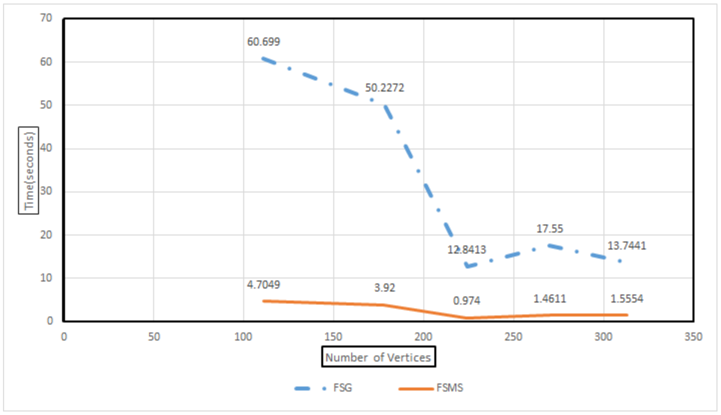


Figure 5: Comparing the performance of FSMS vs. FSG for frequent subgraph mining in graphs with different numbers of labelled vertices and labelled edges. For both algorithms the same minimum support was selected. However we had to keep the minimum support threshold high to be able to compare the performance of FSG and FSMS; this was because FSG was not able to finish the processing for low minimum support.

different values of minimum support were tested; [Table 1] shows FSMS's performance for a representative selection of the graphs. In these chemical datasets, the numbers of distinct vertex labels and distinct edge labels are very low. Hence there exist an extensive number of instances for different sizes of subgraphs in each graph. Because of that, we used relatively high minimum support for our experiments, and still the total number of discovered frequent subgraphs by FSMS and FSG are fairly large as shown in [Table. 1]. Considering the large number of discovered frequent subgraphs and comparing the performance of FSG and FSMS, we can see that the FSMS algorithm requires a short amount of time to discover a large number of graphs without performing isomorphism.

5. FUTURE WORK

Many current graph mining algorithms assume that a graph can be read in to memory and processed on a single, commodity machine. However, that is not always the case; dataset sizes are growing exponentially. According to Cisco, global cloud data will reach

Table 1: Performance of FSMS vs. FSG algorithm on chemical datasets with different graph properties. Columns in this table represent the number of vertices, the number of edges, the number of distinct vertex labels, the number of distinct edge labels, minimum support, total number of discovered frequent subgraphs of different sizes, the maximum size of subgraphs found by FSMS and FSG, and running time of FSMS and FSG in seconds, respectively. A dash in the table means we had to abort the computation for FSG because it was taking too long to complete execution.

Vertices	Edges	Vertex Labels	Edge Labels	Minimum Support	Total # of Frequent Subgraphs	Iterations	FSMS(sec)	FSG(sec)
111	119	3	2	30	360	21	4.70	839.69
111	119	3	2	13	11756	31	97.07	-
178	198	3	2	35	366	35	3.92	344.22
178	198	3	2	13	11840	30	84.64	-
224	252	3	2	52	79349	39	3646.12	-
270	306	3	2	60	6407	30	324.75	-
270	306	3	2	150	15	7	1.46	17.55
313	356	3	2	60	88381	38	4564.31	-
351	402	3	2	70	155973	31	8439.33	-
407	363	3	2	70	169879	31	9144.58	-
432	493	5	2	80	70981	30	5144.84	-
498	567	5	2	70	147645	31	9161.96	-

approximately 715 *EB* per month by the end of 2018 [16]. Technologies to be considered for that purpose include Hadoop, as well as Pegasus [17], a big graph mining system built on top of MapReduce. This will require a redesign of the FSMS algorithm to remove linear dependencies. The distributed implementation of FSMS will be benchmarked against other existing distributed frequent subgraph mining programs such as MiRage [18] to compare its performance. Additionally, we want to find more efficient ways of reading and writing graphs, which we believe will dramatically increase the speed, especially in a distributed implementation. Methods could include compression and/or SQL representation of the graph input and output. We also want to work on improving the memory usage of the algorithm. Depending upon the type of the graph, the number of generated subgraphs could be exponentially large and since all of them are kept in the RAM, FSMS needs some heuristic improvements in order not to run out of memory. Finally, we will investigate graph visualization and visual steering of the FSMS algorithm as discussed in [19]. This could provide a useful option for viewing the subgraphs as they are being found, and adding structural and semantic constraints during the process as suggested in [12].

6. CONCLUSION

Over the past 15 years, several methods have been developed to address the problem of identifying frequent subgraphs, particularly in graph-transaction settings. In this paper, we proposed a Frequent Subgraph mining algorithm, FSMS, that eliminates the process of isomorphism during frequent subgraph mining by using Mapping Sets. We demonstrated the efficiency of FSMS completely and accurately in finding (and listing) all existing frequent subgraphs in some real-world datasets. We are confident that, with additional work, the processing time and scalability of FSMS will be further improved, making it a viable FSM solution in the domains of graph query processing and frequent subgraph mining to discover recurrent patterns in scientific, spatial and relational datasets.

REFERENCES

[1] Google. Inside Search: Algorithms [Online] Written: 2012. Accessed: 04-30-2015.

[2] A. Clement (2014). NSA Surveillance: Exploring the geographies of internet interception. iConference 2014 Proceedings. doi:10.9776/14119, pp. 412-425

[3] Daniel R. Rhodes, Jianjun Yu, K. Shanker, Nandan Deshpande, Radhika Varambally, Debashis Ghosh, Terrence Barrette, Akhilesh Pander, Arul M. Chinnaiyan, ONCOMINE: A cancer microarray database and integrated datamining platform, Neoplasia, Volume 6, Issue 1, January-February 2004, pp. 1-6, ISSN 1476-5586, http://dx.doi.org/10.1016/S1476-5586(04)80047-2. Accessed 04-30-2015.

[4] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. Computer Networks, 33, 2000.

[5] D. A. Bader and K. Madduri. A graph-theoretic analysis of the human proteininteraction network using multicore parallel algorithms. ParallelComput., 2008.

[6] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On powerlaw relationships of the internet topology. SIGCOMM, Aug-Sept. 1999, pp. 251-262

[7] Kuramochi, Michihiro, and George Karypis. "Finding frequent patterns in a large sparse graph*." Data mining and knowledge discovery 11.3 (2005): 243-271.

[8] M. Gholami and A. Salajegheh, A survey on algorithms of mining frequent subgraphs. International Journal of Engineering Inventions 1.5 (2012), pp. 60-63.

[9] A. Inokuchi, T. Washio, and H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data. Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery. Springer-Verlag, 2000.

[10] M. Kuramochi, and G.Karypis. Frequent subgraph discovery. Proceedings of the 2001 IEEE International Conference on Data Mining. IEEE Computer Society, 2001.

[11] X. Yan and J.W. Han. gSpan: Graph-based substructure pattern mining. Proceedings of the 2002 IEEE International Conference on Data Mining. IEEE Computer Society, 2002.

[12] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. GRAMI: Frequent subgraph and pattern mining in a single large graph. Proceedings of the VLDB Endowment, 2014, pp. 517-528.

[13] M. Kuramochi and G. Karypis. GREW - A scalable frequent subgraph discovery algorithm. Proceedings of ICDM, 2004, pp. 439-442.

[14] Lu, Wei, et al. "Efficiently extracting frequent subgraphs using mapreduce." Big Data, 2013 IEEE International Conference on. IEEE, 2013.

[15] National Center for Biotechnology Information. PubChem BioAssay Database; AID=2299, Source=Scripps Research Institute Molecular Screening Center, http://pubchem.ncbi.nlm.nih.gov/assay/assay.cgi?aid=2299 (accessed Feb. 22, 2011).

[16] Cisco. Cisco global cloud index: forecast and methodology 2013-2018 White Paper. [Online]. http://www.cisco.com/c/en/ us solutions/collateral/ serviceprovider/global-cloud-index-gci/Cloud Index White Paper.html. Written 2014. Accessed 04/27/2015.

[17] U. Kang and C. Faloutsos. Big graph mining: algorithms and discoveries. SIGKDDD Explorations. Vol 14, Issue 2, 2013, pp. 29-36.

[18] M. Bhuiyan and M. Al Hasan. MiRage: An iterative MapReduce based subgraph mining algorithm. Jul. 22, 2013, arXiv:1307.5894, accessed 05/31/2015.

[19] K. Puolamiki, P. Papapetrou, and J. Lijffitj. Visually controllable data mining methods. Proceedings of the 2010 IEEE International Conference on Data Mining Workshops. Dec. 2010, pp. 409-417.

II. ALL-INCLUSIVE FREQUENT FREE SUBTREE MINING USING AUTOMORPHISM LIST

Armita Abedijaberi and Jennifer Leopold

Department of Computer Science

Missouri University of Science and Technology, Rolla, Missouri 65401

Graphs are one of the most general formalisms for modeling relationships amongst complex, structured data. Unfortunately, discovering frequently occurring subgraphs in a large graph database comes with high cost due to two computationally expensive operations: (1) checking whether a single graph contains a particular subgraph in order to determine how frequently the subgraph occurs in the larger graph, and (2) determining whether two graphs are isomorphic in order to avoid generating (and hence examining) a candidate subgraph multiple times.

The first problem is an instance of the subgraph isomorphism problem, which is NP-complete. The second problem is an instance of the graph isomorphism problem, which is, at best, a quasi-polynomial time complexity problem. The high complexity of both problems largely stems from the existence of cycles in the graphs.

Herein we present a frequent subgraph mining algorithm for free trees. Unlike other frequent subgraph mining algorithms, our algorithm identifies all instances of frequent subtrees (with no false negatives) rather than just returning the pattern for each frequent subtree found. Additionally, our algorithm is efficiently implemented: (1) it significantly reduces the candidate generation cost whilst avoiding traditional, expensive graph isomorphism tests, and (2) it is designed for distributed computing.

1. INTRODUCTION

With the unrelenting pervasiveness of data interconnectivity, graphs continue to be useful formalisms for modeling relationships amongst complex, structured data [25]. Discovering meaningful patterns in graph datasets often can be achieved through frequent subgraph mining whereby we are interested in subgraphs that occur at least a certain number of times in a larger single graph or a collection of graphs. For example, a social network of terrorists can be identified by examining the number of communications (e.g., emails, text messages, and phone calls) between certain individuals. As another example, the existence of certain chemical structures necessary for the presence of a particular kind of cancer can be identified by modeling proteins as graphs, and counting the frequency of certain subgraphs in positive and negative protein samples [20].

Unfortunately, discovering frequently occurring subgraphs in a large graph dataset comes with high cost due to two computationally expensive operations: (1) checking whether a single graph contains a particular subgraph in order to determine how frequently the subgraph occurs in the larger graph, and (2) determining whether two graphs are isomorphic in order to avoid generating (and hence examining) a candidate subgraph multiple times. The first problem is an instance of the subgraph isomorphism problem, which is NP-complete. The second problem is an instance of the graph isomorphism problem, which is, at best, a quasi-polynomial time complexity problem [1]. The high complexity of both problems largely stems from the existence of cycles in the graphs.

A free tree is a connected, acyclic, undirected graph. It is a specialization of a general graph that has been widely used for applications in natural language processing, networks, bioinformatics, and computer vision [2]. Herein we present a frequent subgraph mining algorithm for free trees. The restriction to free trees avoids many of the undesirable theoretical properties and algorithm complexity incurred by general graphs while still providing a useful tool for a wide range of applications. The novelty and benefits of our algorithm are as follows. Unlike most other frequent subgraph mining algorithms, our algorithm identifies all instances of frequent subtrees (with no false negatives) rather than just returning the pattern for each frequent subtree found. Additionally, our algorithm is efficiently implemented: (1) it significantly reduces the candidate generation cost whilst avoiding traditional, expensive graph isomorphism tests, and (2) it is designed for distributed computing (e.g., Hadoop or Spark).

The organization of this paper is as follows. In Section 2, we provide the definitions of graph terminology that will be used throughout the paper. In Section 3, we briefly discuss background and related work on frequent subgraph mining. Our algorithm for frequent free subtree mining is presented in Section 4, with benchmarked examples and analysis in Sections 5 and 6. Future work, a summary, and conclusions then follow in Section 7.

2. PRELIMINARIES

Labeled Graph: A labeled graph is presented as $G = (V, E, L_V, L_E)$, which consists of a set of vertices V, a set of edges E, and two labeling functions L_V and L_E that associate labels with vertices and edges, respectively. In G, each vertex has a unique id and a label, and each edge has a label; labels of vertices (edges) are not necessarily distinct. Herein we assume that the finite set of labels is built upon alphabet Σ whose elements follow an alphabetical order. We assume that the label function $L_V(L_E)$ facilitates an efficient lookup of all vertices (edges) with a particular label; similarly, we assume that, given a vertex (edge), we can find its label in $O(1)$ time. We will use v_i and l_i to refer to the vertex with id i and label of vertex with id i, respectively. In addition, $e_{i,j}$ and $l_{i,j}$ will refer to the edge between v_i and v_j and its label, respectively. A graph can be directed or undirected. If it is directed then the direction of an edge between two vertices can be considered as one of the properties of the label of that edge. In discussions where distinction between vertex (edge) ids is not important, we will refer to vertices (edges) simply by their labels. We assume that graphs do not contain multi-edges (i.e., two or more edges that are incident to the same two vertices) or self-loops (i.e., an edge from a vertex to itself).

Tree: A tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any acyclic connected graph is a tree. A subtree with k number of edges is denoted as a k-subtree. Tree mining problems can be broadly classified into ordered (or unordered) and rooted (or unrooted) subtree discovery [3]. A rooted unordered tree is a tree where a special vertex, which is often referred to as the root, is distinguished among all vertices. A rooted ordered tree is a tree where the root is distinguished and the order among children of each vertex is explicitly predefined.

Embedded Subtree: A tree $T = (V_t, E_t, L_{V_t}, L_{E_t})$ is an embedded subtree of $G = (V, E, L_V, L_E)$ if (1) $V_t \subseteq V$, (2) $E_t \subseteq E$, (3) $L_{V_t}(v) = L_V(v)$ and $L_{E_t}(e) = L_E(e)$ for all $v \in V_t$ and $e \in E_t$, and (4) $(v_1, v_2) \in E_t$, v_1 is an ancestor of v_2 in G, which is preserved in T [Figure 1].

Induced Subtree: A tree $T = (V_t, E_t, L_{V_t}, L_{E_t})$ is an induced subtree of $G = (V, E, L_V, L_E)$ if (1) $V_t \subseteq V$, (2) $E_t \subseteq E$, (3) $L_{V_t}(v) = L_V(v)$ and $L_{E_t}(e) = L_E(e)$ for all $v \in V_t$ and $e \in E_t$, and (4) $(v_1, v_2) \in E_t$, v_1 is the parent of v_2 in G, which is preserved in T [Figure 1].

Tree Isomorphism: Two labeled trees T and T' are isomorphic, denoted as $T \cong T'$, if there is a one-to-one structure-preserving bijection between T and T', where this bijection preserves the vertex and edge labels. Two trees related by isomorphism differ only by the ids of the vertices and edges. Otherwise, there is a complete structural equivalence between two such trees.

Support and Frequent Subtree: Let I_S be the set of isomorphisms of a subtree T in graph G. Given a minimum support threshold τ, the frequent subtree mining problem is to find all subtrees T in G such that $|I_S| \geq \tau$. The advantage of limiting frequent subtrees

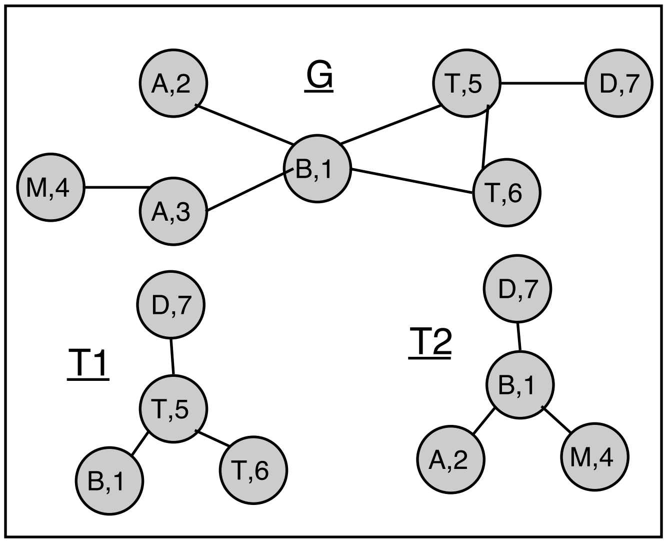


Figure 1: Example of an induced subtree $T1$ and an embedded subtree $T2$ of graph G. Each vertex is shown with its label (an uppercase alphabetic character) and its unique id (a positive integer). An induced subtree preserves the parent-child relation between two vertices. An embedded subtree preserves the ancestor relations but not necessarily the parent relations for vertices in G.

to only those with disjoint edges is computational tractability [4]. But this comes at the expense of disregarding potentially useful information. Consider the example shown in [Figure 2]. If only edge-disjoint isomorphic subtrees are considered, the vertex labeled A will not be included in any frequent subtrees when minimum support is ≥ 2. However, if we allow isomorphic subtrees to share edges, then the subtrees containing vertices labeled T, B, and A will have frequency 2 with instances, $\{e_{2,3}, e_{3,1}\}$ and $\{e_{4,3}, e_{3,1}\}$.

Two widely used frequency notions are per-subtree frequency, where only the occurrence of a subtree pattern is important; and per-occurrence frequency, where the exact number of occurrences of a subtree pattern is important too. Per-occurrence frequency counting is computationally more expensive than per-subtree frequency counting, but it can deliver very important information. To eliminate the cost of computation for generating the candidates an efficient algorithm is required. A graph dataset can consist of a single graph or a collection of transaction graphs $G =< G_1, G_2, ..., G_n >$. The work presented in this paper is

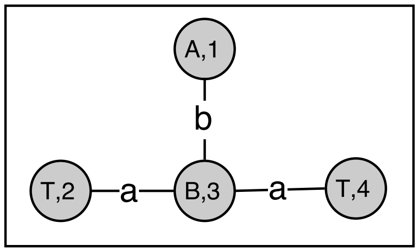


Figure 2: Example of a graph where all vertices and edges are labeled and each vertex has a unique id as well. When minimum support is ≥ 2, a particular vertex (i.e., v_1) will only be included in a frequent subtree if we allow isomorphic subtrees to share edges (i.e., $(e_{1,3})$).

focused on finding all frequent unordered, unrooted, induced subtrees (i.e., free subtrees) in a single graph, but could easily be modified to be applicable on a transaction graph setting.

3. BACKGROUND AND RELATED WORK

In this section, we briefly discuss related work on frequent subtree mining. Numerous frequent subgraph mining algorithms have been developed; see [5] for a comprehensive survey. The graph theory based approaches typically start by generating and examining small candidate subgraphs (e.g., a single vertex), then expand frequently occurring subgraphs by a single edge and/or vertex to generate larger candidates. The algorithmic approaches primarily differ by whether the search space is explored depth-first (pattern growth) or breadth-first (Apriori). Frequent subgraph mining algorithms are also categorized based on whether they are intended for a single graph [36] or a transaction graph (i.e., collection of graphs) setting [23], and whether or not they produce only approximate results [8]. In all of the well-known subgraph mining algorithms, the results that are returned are the patterns of the frequent subgraphs that satisfy the minimum occurrence threshold; many algorithms do not return the actual number of occurrences for each frequent pattern found and no algorithms report all instances of frequently occurring patterns. Depending upon the application, the latter piece of information could be important to know. For example, a chemist might want to know which different patterns seem to occur in the same subgroup of effective medicines or on the other hand which patterns occur in different subgroups of effective medicines. So, they will need to know all the occurences of such patterns.

Modifying one of the existing, well-known frequent subgraph mining algorithms so that it returns that information could have devastating repercussions on its time complexity, as most such algorithms terminate searching once the number of occurrences of a subgraph has met the minimum threshold; they do not continue to look for all occurrences of that subgraph. With the advent of World Wide Web and the need for mining semi-structured XML data, free trees have been studied intensively in various areas. A free tree is a generalization of linear sequential patterns, so it preserves plenty of structural information of the datasets to be mined. At the same time, it avoids undesirable theoretical properties and logarithmic complexity incurred by graph computations. As the middle ground, a free tree provides us with a good compromise between the more expressive yet computationally harder general graph and the faster yet less expressive path in data mining research [2].

The work herein is restricted to finding frequent free trees rather than general graphs. Thus we specifically mention two existing frequent subtree mining algorithms: FT discovers frequent subtrees in a tree database [15], [6], [16], [17], and FG finds frequent subtrees in a graph database [18], [7]. FT is an Apriori-based algorithm. In each step, FT generates candidate frequent subtrees in a breadth-first manner. That is, in level $n+1$, all candidates of frequent $(n+1)$-subtrees are generated using frequent size n-subtrees found in level n, $n \geq 1$, where n is the number of edges in the subtree. FT follows a pattern-join approach to generate candidate frequent subtrees. Two frequent n-subtrees which share the same frequent $(n-1)$-subtree, which is referred to as the core, are joined to generate a candidate $(n+1)$-subtree. The Apriori-based algorithms are not likely to achieve a good performance because the exponential growth of potential frequent subtrees inevitably requires a considerable memory consumption in the mining process [2]. FG is a depth-first mining algorithm which follows the pattern-growth strategy to generate candidate frequent subtrees. Given a frequent n-subtree, FG first counts its frequency by traversing the graph database. At the same time, all vertices at the lowest layer of that tree are selected as extension points to generate candidate trees of size $n+1$. The advantage of the depth-first mining algorithm FG over the Apriori-based algorithm FT is its relatively small memory consumption. In order to enumerate candidate frequent trees of size $n+1$ in FT, all frequent size n trees must be held in memory. The large memory consumption costs a great number of physical page swaps between main memory and disk. However, there still exist several disadvantages for FG. First, the candidate generation process to grow vertices on the extension points of a frequent tree may generate redundant candidates, which incurs repetitive computations if no pre-pruning is provided. Second, during each database scan for frequency counting, all occurrences of a tree in the graph database must be computed from scratch, if no further optimization techniques are provided. In general these two algorithms are slightly more efficient than frequent subgraph mining algorithms because of their restriction to free trees. However, they still suffer from one of the same limitations of the graph algorithms: they do not report all instances of the frequent patterns they discover.

4. METHODOLOGY

In this section we present Frequent Free Subtree Mining algorithm using Automorphism List (SMAL). The concepts we discuss are illustrated with examples for further clarification.

4.1. SUBTREE REPRESENTATION

A frequent subtree with k edges is denoted as a k-subtree. The vertices in a frequent k-subtree T are uniquely and consecutively indexed according to the order that they have been added to T (i.e., the index of each vertex represents its location in the subtree). The index of the first and last added vertex in T are 1 and $k+1$, respectively. A k-subtree is represented as a list of edges which is referred to as an automorphism list. Each edge is represented by a pair of ids of vertices at two ends of it.

In a subtree there is no cycle meaning that each vertex can have only one parent. However, a vertex in a subtree can have more that one child. Each edge as a pair represents the (parent,child) relationship of its ending vertices in the subtree. In an automorphism list of a subtree, a vertex id can appear as the right element (child) in a pair only once (i.e., a vertex can only have one parent). However, it can appear as the left element (parent) in a pair more than one time (i.e., a vertex can have more than one child). On that note, in order to find the index of a vertex in a subtree we only need to find the location of the pair in the automorphism list where that vertex id appears as the right item .

Consider the example in [Figure 3]. $T1$: $\{B-b-T\}$ is a 1-subtree with two vertices and one edge, where each vertex is assigned an index. The procedure of index assignment to a vertex of a subtree will be fully explained later in this paper. In $T1$ a vertex with label B and index 1 is connected to another vertex with label T and index 2 through an edge with label b. For each subtree we assume that there is an imaginary root with index 0 and id -1, and vertices with index 1 and 2 are connected to the root instead of being connected to each other. We also assume that none of the vertices in the dataset has id of -1, otherwise we must choose an id for the root that does not exist in the graph.

In [Figure 3] $T1$ has two instances (i.e., $e_{1,4}$ and $e_{6,4}$) in graph G. The automorphism list of $T1$ contains two lists, one with respect to each instance of $T1$. The number of pairs in each list represents the number of vertices in the subtree. Subtree $T2$ is a 2-subtree in G with two instances (i.e., $[e_{3,1}, e_{1,4}]$ and $[e_{5,6}, e_{6,4}]$). Likewise, $T2$ has an automorphism list containing two lists where each list contains pairs of edges in an instance of $T2$.

The automorphism list of $T1$ with two instances is $[[(-1,1),\ (-1,4)],$ $[(-1,6),(-1,4)]]$. As mentioned before, in the list of an instance, the location of a pair

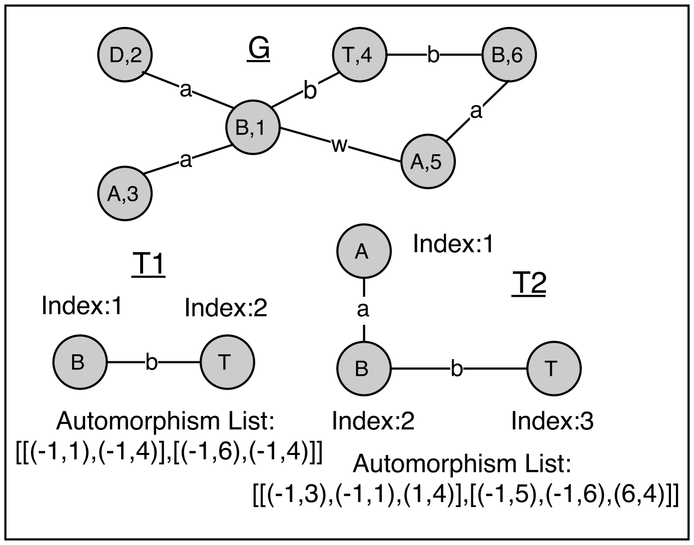


Figure 3: Example of a 1-subtree ($T1$) and a 2-subtree ($T2$) in graph G. Each subtree is represented with an automorphism list that contains lists of instances (in terms of vertex ids) for that subtree in graph G.

in which a vertex appears as the second (right) item is considered the index of that vertex in that instance. For example, in the first instance, the vertex with id 1 appears as the second item in the pair $(-1,1)$ and $(-1,1)$ is the first pair in the list, so the index of 1 is 1. Similarly, the vertex with id 4 appears as the second item in the pair $(-1,4)$ and $(-1,4)$ is the second pair in the list so the index of 4 is 2.

The automorphism list of $T2$ with two instances is equal to $[[(-1,3),(-1,1),(1,4)],[(-1,5),(-1,6),(6,4)]]$. In the first instance, vertices 3, 1, and 4 have the index of 1, 2, and 3, respectively. Similarly, in the second instance, vertices 5, 6, and 4 have the index of 1, 2, and 3, respectively.

4.2. EXTENSION LABEL

SMAL is an iterative algorithm that employs a pattern growth-based approach whereby it starts by examining small candidate subtrees, and then iteratively generates and tests larger candidate subtrees. While mining a graph G, in order to generate a candidate

k-subtree T', a frequent $(k-1)$-subtree T is extended by adding a new edge, hence a new vertex, to it. Due to the acyclicity property of trees, an edge addition between an existing vertex and a new vertex is only allowed. Adding a connection between two already existing vertices of a tree creates a cycle.

Assume that v_i is a vertex of T that is extended by v_j, which is one of its neighbors in G. For this extension, the extension label will be represented as $< l_{ij}, l_j >$, where l_{ij} represents the label of the edge between v_i and v_j and l_j is the label of v_j. Each vertex can have multiple neighbors with the same extension label or different extension labels [Figure 4].

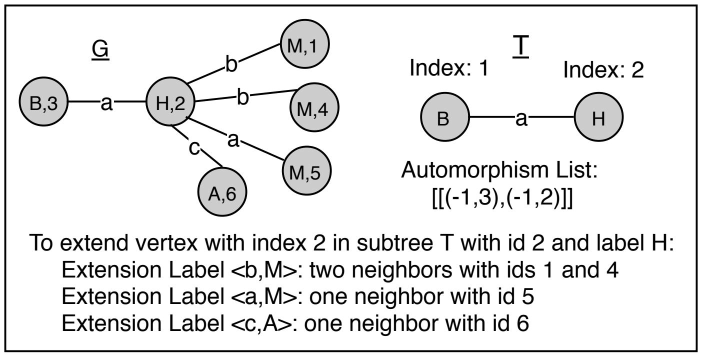

G
B,3
a
H,2
b
M,1
b
M,4
a
M,5
c
A,6
T
Index: 1
Index: 2
B
a
H
Automorphism List:
[[(-1,3),(-1,2)]]
To extend vertex with index 2 in subtree T with id 2 and label H:
Extension Label <b,M>: two neighbors with ids 1 and 4
Extension Label <a,M>: one neighbor with id 5
Extension Label <c,A>: one neighbor with id 6

Figure 4: Example of extension labels for a vertex in a subtree. Subtree T can be extended by three different extension labels through v_2.

4.3. AUTOMORPHISM LISTS

A vertex bijection between two isomorphic trees T and T' is a one-to-one correspondence between the elements of two vertex-sets $f : V_T \longrightarrow V'_T$, where every element of one set is paired with exactly one element of the other set, and every element of the other set is paired with exactly one element of the first set; there are no unpaired elements. This kind of bijection is also known as a structure-preserving bijection.

An automorphism of a tree T is an isomorphism of T with itself $f : V_T \longrightarrow V_T$ (i.e., a mapping from the vertices of T back to vertices of T such that the resulting tree is isomorphic with T). In fact, an automorphism of a tree is a form of symmetry in which the tree is mapped onto itself while preserving the edge-to-vertex connectivity. The set of automorphisms of T defines a permutation group known as T's automorphism group. The length of such an automorphism group specifies the number of existing vertex bijections from T to T.

Assume that two subtrees T_1 and T_2 are isomorphic. Thus, there exists at least one bijection $f : V_{T_1} \longrightarrow V_{T_2}$. If T_1 is isomorphic to itself (which implies T_2 is isomorphic to itself too), there exists L number of bijections between T_1 and T_2, where L is equal to the length of the automorphism group of T_1 (or T_2) times the number of isomorphic trees (which in this case is 2).

In SMAL, for each subtree T there is an automorphism list that contains the vertex bijection between the vertices of all instances of T in a graph G. If T is a subtree that is isomorphic to itself, then all the possible bijections between the vertices of all instances are stored within the automorphism list. As shown in [Figure 5], subtree T has two instances (i.e., $[e_{2,1}, e_{2,3}]$ and $[e_{6,7}, e_{6,5}]$) in graph G. Subtree T is isomorphic to itself and there exist two vertex bijections between its vertices. The vertex bijections $B1$ and $B2$ show two possible one-to-one correspondences between vertices of each of the instances of T. In addition, the vertex bijection $B3$ shows two possible one-to-one correspondences between vertices of the two instances of T. Therefore, with two instances there exist four vertex bijections that are stored in the automorphism list of T.

4.4. SMAL ALGORITHM

The SMAL algorithm can be applied to a directed or undirected graph $G = (V, E, L_V, L_E)$. SMAL is an iterative algorithm that consists of k iterations in order to find all frequent subtrees of different sizes from 1 to k in G. In essence, during the procedure of

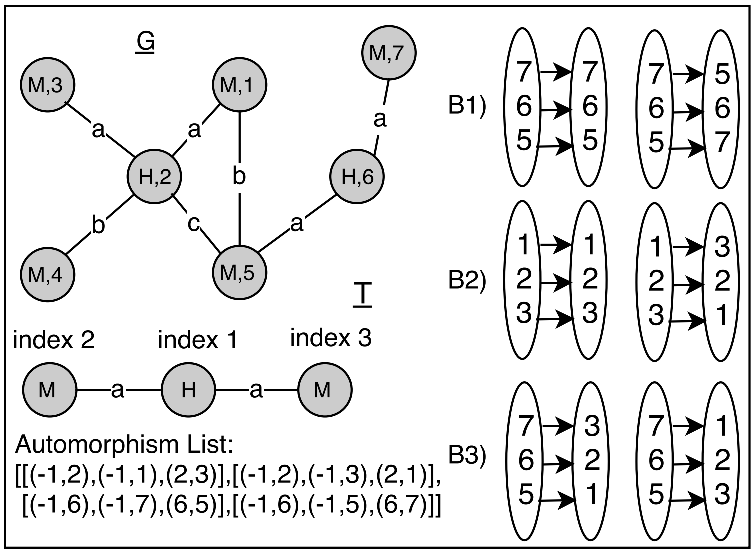


Figure 5: Example of a subtree T that has an automorphism (i.e., T is isomorphic to itself). T has two instances in G where each instance has two sets of automorphism. Hence, the automorphism list of T contains 4 lists.

subtree mining, by the end of iteration i, instances of frequent subtrees of size i are found and candidate subtrees of size $i+1$ will be built from them. Calculation of frequency of a subtree candidate would normally involve subtree isomorphism which is eliminated by SMAL by using an automorphism list. The idea is based on the fact that when two or more isomorphic subtrees expand from the equivalent vertices (in terms of bijections between isomorphic subtrees) with same-labeled vertices and same-labeled edges, the extended subtrees will be automatically isomorphic to each other. That is, for a frequent subtree T, all the vertices with the same index in each instance of the automorphism list are equivalent to each other. So, if equivalent vertices extend by similar extension labels, the extended subtrees will be automatically isomorphic to each other. Hence, there is no need to apply any additional tests for isomorphism.

The SMAL algorithm starts with finding the frequent edges in the graph G as frequent subtrees of size one, where each frequent 1-subtree is represented by an automorphism list. During each iteration, candidate subtrees are generated from frequent subtrees one size smaller. For instance, a candidate $(i+1)$-subtree T' is generated by iterating through the automorphism list of a frequent i-subtree T. If the number of occurrences of T' in G (i.e., frequency of T') is greater than or equal to the user-defined frequency threshold τ, T' is considered frequent and during the next iteration of SMAL it will be used to generate candidate subtrees of larger size. Otherwise, T' will be discarded. For each index k in the automorphism list of T, new candidate subtrees are formed as follows: the extension labels for all the vertices with index k in instances of T are generated and grouped together in a form of (extension label, values), where values are the automorphism list of new candidates for each extension label. The way that the automorphism list for a candidate is generated is by appending the edge pair (current vertex id (parent), added neighbor id (child)) to the current automorphism list. As mentioned previously, since all vertices with the same index have a similar role in T (i.e., equivalent to each other), if they extend with the same extension label, all the extended subtrees will be automatically isomorphic to each other. Each group of extension labels can lead to a distinct candidate subtree.

As an example, consider [Figure 6] and assume that $\tau = 2$. $T1$, $T2$, and $T3$ are three frequent edges in the graph G. For each frequent 1-subtree an automorphism list is generated as follows: each edge has two vertices, and we assign the index 1 to the vertex with the smaller label and index 2 to the vertex with the larger label. In $T1$ the vertex with label A has index 1 and the vertex with label B has index 2. If both vertices have the same label (i.e., the edge has an automorphism) (e.g. $T3$), then we need to keep both automorphisms of that edge; the first time the first vertex has index 1 and the second vertex has index 2, and the second time the first vertex has index 2 and the second vertex has index 1. Then, according to the assigned indexes and vertex ids of the instances of a frequent edge, the automorphism list for each 1-subtree is formed.

In this example, $T1$ has 4 instances in G including $e_{1,2}$, $e_{1,3}$, $e_{7,8}$, and $e_{7,9}$, so it has four edge-lists in its automorphism list. Vertices with label A have index 1 and vertices with label B have index 2. We assume that there is an imaginary root for each subtree with id -1 that is the parent of the first two vertices of that subtree, and those two vertices are connected to the root instead of being connected to each other. Hence, the edge-lists for $e_{1,2}$, $e_{1,3}$, $e_{7,8}$, and $e_{7,9}$ are $[(-1,1),(-1,2)]$, $[(-1,1),(-1,3)]$, $[(-1,7),(-1,8)]$, and $[(-1,7),(-1,9)]$, respectively. Finally the automorphism list of $T1$ is a list that contains the edge-list of all of its instances.

Subtree $T3$ has 2 instances in G including $e_{2,3}$ and $e_{8,9}$. Since the label for both vertices is the same, we keep two edge-lists per instance. So, $T3$ has four edge-lists in its automorphism list. The edge-lists for $e_{2,3}$ are $[(-1,2),(-1,3)]$ and $[(-1,3),(-1,2)]$. Similarly, the edge-lists for $e_{8,9}$ are $[(-1,8),(-1,9)]$ and $[(-1,9),(-1,8)]$. Finally the automorphism list of $T3$ is a list that contains the edge-list of all of its instances.

To generate candidate 2-subtrees from the vertex with index 2 (vertex with label C) in $T2$, we find the extension labels for the vertices v_{10}, v_{12} and group them together. v_{12} has one extension label of $\langle d,L \rangle$ (vertex with id 14) and v_{10} also has one extension label of $\langle d,L \rangle$ (vertex with id 11). The automorphism list for $T2$ is $[[(-1,8),(-1,10),(10,11)], [(-1,3),(-1,12),(12,14)]]$ and is considered frequent. Since $T5$ is built by adding a new vertex to an existing vertex of $T2$ and the last index is 2, the new added vertex has the index of 3.

Following the above procedure will create all the automorphism groups of a subtree. The automorphism list of $T4$ has 4 edge-lists. Keeping the automorphism list helps to find the bijections of a subtree and avoid generating duplicated candidates for the following iterations; this will be explained in the following sections. However, it requires the algorithm to walk through the edge-lists and count the distinct ones to determine whether or not a candidate is frequent.

4.5. HOW TO AVOID FALSE NEGATIVES

A false negative occurs when a frequent subtree mining algorithm fails to generate some candidate subtrees; this can lead to missing potential frequent subtrees. There are

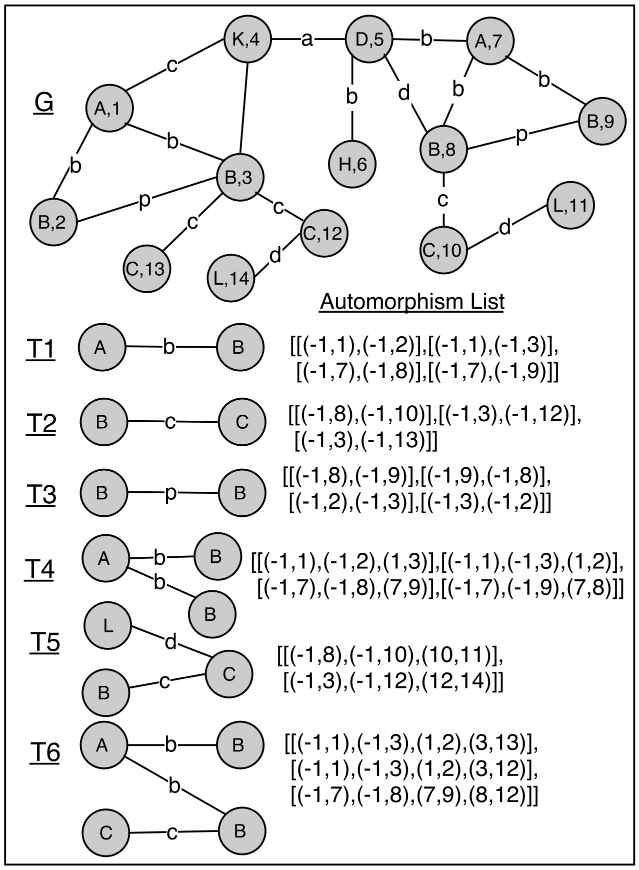


Figure 6: A running example of SMAL.

common approaches practiced by well known frequent subgraph/subtree mining algorithms that can result in false negatives. These approaches are discussed below.

4.5.1. Removing the Infrequent Vertices/Edges. Some frequent subgraph mining algorithms remove the infrequent vertices/edges to reduce the search space to a smaller one in order to speed up the process of mining [11], [12]. However, doing so can lead to failure in finding the potential frequent subgraphs (subtrees). As an example in [Figure 7], graph G with 5 vertices and 4 edges has one frequent edge ($A-s-B$) and two infrequent edges ($B-b-H$ and $H-s-A$) when $\tau = 2$. By removing the infrequent edges from G, a frequent subtree mining algorithm will only detect and report $T1$ as a frequent 1-subgraph and stop. However, by keeping the infrequent edges in the graph, more larger subtrees (i.e.,

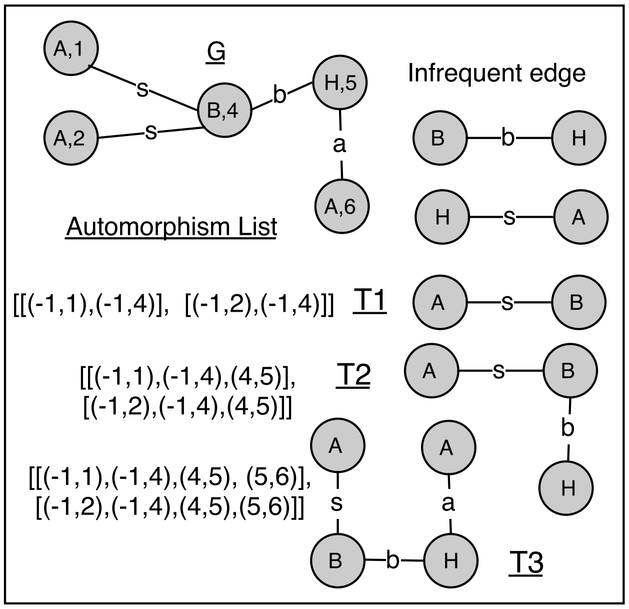


Figure 7: An example to show that keeping the infrequent edges results in detecting larger frequent subtrees that would have been missed otherwise. In this example, if infrequent edges get removed from G, then $T2$ and $T3$ will not be detected as frequent subtrees.

$T2$ and $T3$) will be detected as frequent. In fact, these larger subtrees have anti-monotonic support [8] of 2 in G.

Anti-monotonic support happens when the support of a pattern is larger than the support of one of its sub-patterns. In fact, the anti-monotonicity happens when the occurrences of a subgraph (subtree) are not independent (i.e., they overlap). One common reason for overlapping of occurrences of a subtree is due to the existence of junction vertices in the graph. In [Figure 7], v_4 is called a junction vertex. A junction vertex is a vertex with two or more similar extension labels. In other words, same-labeled vertices (via same-labeled edges) are connected to a junction vertex. For instance, v_4 has 2 neighbors (v_1 and v_2) with extension label $\langle s,A \rangle$). Existence of such a structure in a graph causes missing potential frequent subtrees with anti-monotonic support if infrequent edges are removed. To avoid any false negatives, SMAL does not prune the graph by removing infrequent vertices/edges.

4.5.2. Removing the Examined Frequent Edges. Similar to removing infrequent edges from a graph, removing the already examined frequent edges is another common mistake that is carried out in some frequent subpattern algorithms [13], [14], [8]. In practice, many pattern-growth algorithms start with one frequent edge and expand it to find/test all candidate patterns of different sizes. Then, the frequent edge that is already examined is removed from the graph to make the search space smaller. After that, the next frequent edge will be used for expansion. However, this intuitive approach can result in the situation of false negatives. Consider the example in [Figure 9], There are two graphs, $G1$ and $G2$, with slightly different structures. Note that only some of the frequent subtrees of $G1$ and $G2$ are presented in the picture. With minimum support of 2, both $G1$ and $G2$ have the same frequent 1-subtrees (i.e., $A-s-B$ and $K-a-C$). For $G1$, starting with the frequent edge $K-a-C$, frequent subtrees $T2$, $T4$, and $T5$ are found. Then, we remove $K-a-C$ from $G1$ and start with the next frequent edge (i.e., $A-s-B$). The frequent subtrees $T1$ and $T3$ will be found. For $G2$, starting with the frequent edge $K-a-C$, frequent subtree $T2$ is found. Then, we remove $K-a-C$ from $G2$ and start with the next frequent edge (i.e., $A-s-B$). The frequent subtrees $T1$ and $T3$ will be found, but the frequent subtrees $T4$ and $T5$ will be missed.

4.6. HOW TO AVOID DUPLICATE CANDIDATES

SMAL is designed to run on a single input graph and return all its frequent subtrees along with their occurrences using a pattern-growth approach. Typically, the bottleneck of a pattern-growth approach is the duplication of generated candidates. One single template can be generated starting from different frequent edges. To reduce the cost of computation, it requires that we not generate duplicate candidates.

4.6.1. Expansion from Left to Right. An unordered tree may appear as different ordered trees, which are considered isomorphic to each other. For instance, in [Figure 8] subtrees $T4$ and $T5$ are isomorphic to each other even though the vertices are in different order. To avoid duplication, it is crucial to let a subtree grow in a systematical way such that no previously generated candidates get generated again.

As previously mentioned, a candidate $k+1$-subtree is generated by adding an edge to a k-subtree. A k-subtree has k edges and $k+1$ vertices where all of its vertices are indexed from 1 to $k+1$. SMAL uses a leg attachment technique to avoid generation of duplicated candidates. An extension happens as follows:

1. In order to extend a frequent k-subtree T to generate a candidate $k+1$-subtree T', a new vertex must get connected to one of the $k+1$ vertices of T. Each of the vertices in T is assigned an index from 1 to $k+1$. Let's assume that the last added edge of T is a connection between vertices with indexes i and $k+1$. A new vertex can get connected to an existing vertex with index greater than or equal to i. In simple words, a vertex with index j cannot be extended where there is a connection between vertices indexed larger than j in a subtree. This technique makes sure that a subtree grows from left to right and top to bottom, and is a necessary condition for avoiding duplicate candidates. Hence, in any generated subtrees the indexes of vertices is increasing from left to right, and indexes of children are larger than indexes of their parents. In [Figure 8], the indexes are shown next to the vertices of each subtree. We assume that each of these subtrees is a part of a larger subtree. We will refer to each vertex using its index.

In $T3$, $i < j < j+1 < k < k+1 < k+2$. The last added vertex has an index of $k+2$ and the last connection is between vertices with indexes $j+1$ and $k+2$. According to the first rule of leg attachment, in the next iteration of the algorithm, $T3$ can only be extended via $j+1$, $k+2$, k, and $k+1$.

2. A new vertex (child) c with an extension label can be added to an existing vertex (parent) p of a subtree if the extension label of c is greater than or equal to the extension label of its immediate left sibling (i.e the last added child of p). Additionally, the extension label of c, as the n^{th} child of p, must be greater than or equal to the extension label of the n^{th} child of the immediate left equivalent sibling of p. If the immediate left equivalent

sibling of p does not have an n^{th} child, then c cannot be added to v. In $T4$, j and $j+1$ are equivalent to each other. Hence, the last connection, which is $j+1$ and $k+5$, is invalid because the extension label of $k+5$, as the 3^{rd} child of $j+1$ is not greater than or equal to the extension label of the 3^{rd} child of j. On the same note, the last connection in $T2$ is invalid because the extension label of $j+1$ is not greater than or equal to the extension label of j. Similarly, the last connection of $T6$ is considered invalid because j does not have a 3^{rd} child, so $j+1$ cannot have a 3^{rd} child. Examples of valid subtrees are depicted in [Figure 8].

If the new vertex c does not have an immediate left sibling and its parent p does not have an immediate left equivalent sibling, the extension is safe to be added. Finding the immediate left equivalent sibling is trivial; as a subtree grows from left to right the vertices that share a parent (i.e., they are siblings) have consecutive indexes.

3. As mentioned earlier, if a subtree has an automorphism, all of its automorphism groups will be generated automatically. For single-edge subtrees, automorphism testing is trivial and the automorphism list is generated for all 1-subtrees that have an automorphism. After that, as subtrees expand, the automorphism lists will be generated automatically. When there is duplication in the automorphism list of a subtree, it means that a subtree has an automorphism. Having the list of all automorphism groups of a tree facilitates finding the equivalent indexes of that subtree. The final condition is that if indexes i, j, k are equivalent to each other, the graph can only be extended from the smallest index. For example, $T1$ has an automorphism, and j and $j+1$ are equivalent vertices. $T1$ can only grow from the vertex with the smallest index in a group of equivalent vertices.

The above conditions are conjunctive. This guarantees there will be no false negatives and no duplicate candidates.

4.6.2. Keeping the Examined Frequent Edges. Consider the example in [Figure 9]. For $G1$, starting with the frequent edge $K-a-C$, frequent subtrees $T2$, $T4$, and $T5$ are found. Then, starting with the next frequent edge $A-s-B$, while keeping the

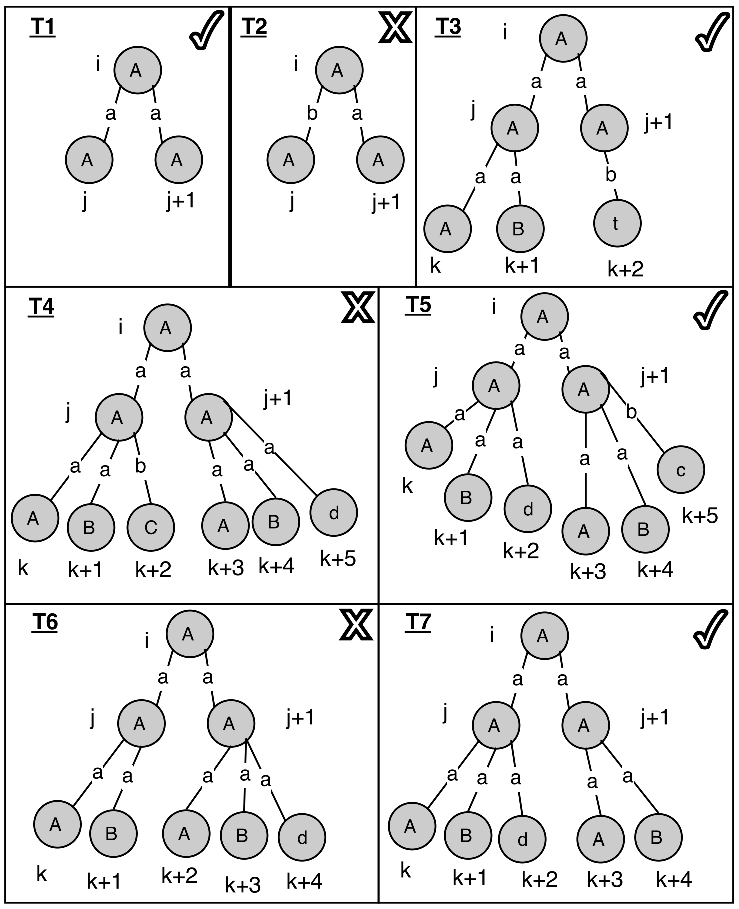


Figure 8: Examples of valid ✓and invalid × connections

instances of $K-a-C$ in $G1$, frequent subtrees $T1$, $T3$, and $T5$ will be found. Subtree $T5$ will be generated two times and is considered a duplicate candidate. As explained previously, removing the examined frequent edges could lead to missing a frequent subtree. Yet keeping them could lead to generating duplicate candidates due to the positions of junction vertices in the input graph.

To address this problem, we do not remove any frequent edges even after that edge is processed. However, to avoid any duplication, after generating $T2$, $T4$, $T5$, $T1$, and $T3$ in $G1$, we only allow $T3$ to expand via vertex C with label extension $\langle a,K\rangle$ (i.e., the added edge in the subtree will be $C-a-K$ which is a previously examined frequent edge), if the number of distinct neighbors of v_4 with extension label $\langle a,K\rangle$ is less than τ. Otherwise, the expansion will generate duplicate candidates. In this case, the number of distinct neighbors of v_4 with extension $\langle a, K \rangle$ is equal to 2 which is not less than τ. Hence, we do not let $T3$ extend via its vertices with index 3 and label extension $\langle a, K \rangle$.

For $G2$, starting with the frequent edge $K-a-C$, a frequent subtree $T2$ is found. Then, starting with next frequent edge $A-s-B$ (while keeping the instances of $K-a-C$ in $G2$), the frequent subtrees $T1$ and $T3$ will be found. We only allow $T3$ to expand via vertex K with label extension $\langle a, K \rangle$ (i.e., the added edge in the subtree will be $K-a-C$, which is a previously examined frequent edge), if the number of distinct neighbors of v_5 with extension label $\langle a, C \rangle$ is less than τ. In this case, the number of distinct neighbors of v_5 with extension $\langle a, C \rangle$ is equal to 1 which is less than τ. Hence, we let $T3$ extend via its vertices with index 3 and label extension $\langle a, C \rangle$, which generates the next frequent subtree $T4$. Similarly, $T4$ is extended through its vertices with index 4 and label C by label extension $\langle a, K \rangle$, because there are fewer distinct neighbor ids than τ. Hence $T5$ is also generated and reported as a 4-frequent subtree.

5. DISTRIBUTED COMPUTATION IN SPARK

In this section we present the distributed implementation of SMAL using the GraphX framework. GraphX is Apache Spark's API for graphs and graph-parallel computation [9]. Spark processes data in-memory and uses "lazy evaluation" to form a Directed Acyclic Graph (DAG) of consecutive computation stages. In this way, the execution plan can be optimized, e.g. to minimize shuffling data around. At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster. The main abstraction Spark provides is a Resilient Distributed Dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel and can automatically recover from node

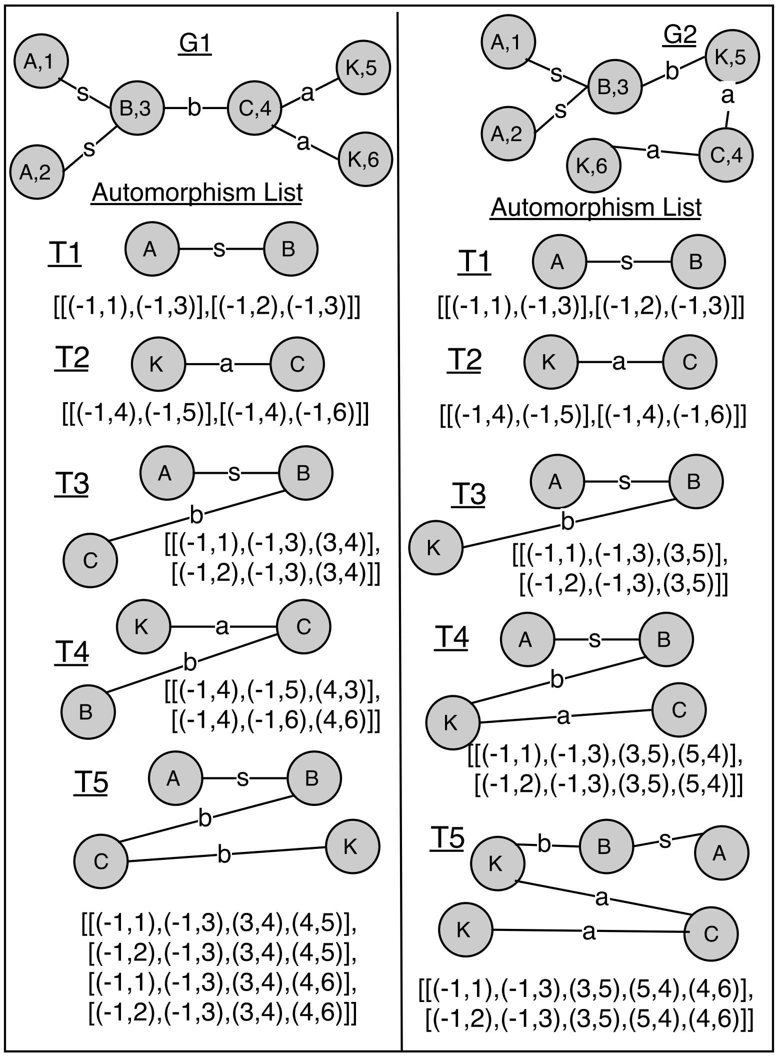


Figure 9: An example to show that keeping the instances of frequent edges even after they are processed results in detecting larger frequent subtrees, which would have been missed otherwise.

failures. RDDs are created by starting with a file in the Hadoop Distributed File System (HDFS) [10]. [Figure 10] shows the workflow of the prototype implementation of SMAL in GraphX. At the beginning, data are stored in a HDFS database, and are read in to create EdgeRDD and VertexRDD to finally generate GraphRDD, which is our search space. Frequent edges will be generated using the *groupBy* method and each frequent 1-subtree will be assigned a global id which is shared between instances of each subtree using the *zipWithIndex* method. Assigned global ids are distinct for each group of subtrees. Next, we apply the *map* method on each subtree to generate candidate subtrees of one size larger using the SMAL algorithm. Here candidates are in the form of pairRDDs with a triple key of (global id, index, extension label), where global id is the global id of the parent subtree, index is the index of vertices in the subtree that has been extended, and extension label is the extension label of an added vertex. A *reduceByKey* method is executed on candidates to group them together. The *reduceByKey* method first groups the candidates of each partition and then groups the partitions to ensure minimum shuffling in the data. Then a *filter* method is executed to discard infrequent subtrees considering the value of τ. Again, a global id will be assigned to each group of frequent subtrees and they are ready to be used in the next iteration. After each iteration, the frequent subtrees are written to HDFS.

6. EXPERIMENTAL EVALUATION

In this section, we discuss our experimental evaluation of SMAL and its distributed version. All experiments were conducted on a Linux (Ubuntu 14.4) machine with 4 cores running at 3.40GHz with 32GB RAM. The experiments were performed on two real graph datasets: CiteSeer and Citation. CiteSeer is a graph that represents publications and citations where vertices are labeled with the computer science area of the publication and edges are labeled with the measure of similarity between corresponding pairs of publications. Citation is a graph that models publications and citations where vertices are labeled with the document id and edges are labeled with the citation relation. The main characteristics of these graphs are summarized in [Table 1].

The minimum support threshold τ is the key evaluation metric as it determines when a subtree is frequent. Decreasing τ results in an exponential increase in the number of possible candidates.

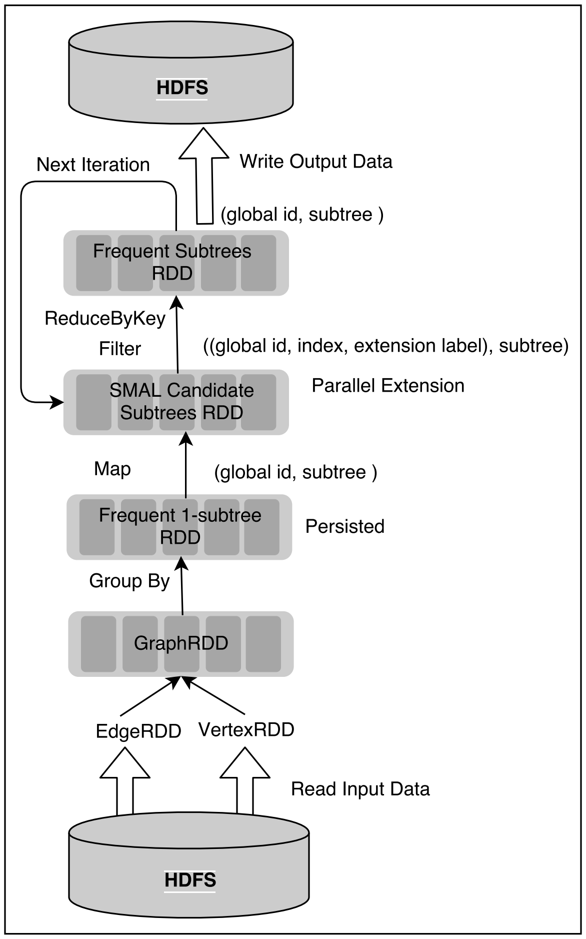


Figure 10: Workflow of Spark Implementation of SMAL

Table 1: Datasets and their characteristics

Dataset	#Vertices	Distinct vertex labels	#Edges	Density
CiteSeer	3312	6	4732	Medium
Citation	29014	742	81353	Sparse

[Figure 11] and [Figure 12] show the results of running SMAL and the Spark version of it (S-SMAL) on CiteSeer and Citation, respectively. In both the figures the red line represents the performance of SMAL and the green line represents the performance of S-SMAL, where the shared x-axis is the minimum support and the left y-axis is the elapsed

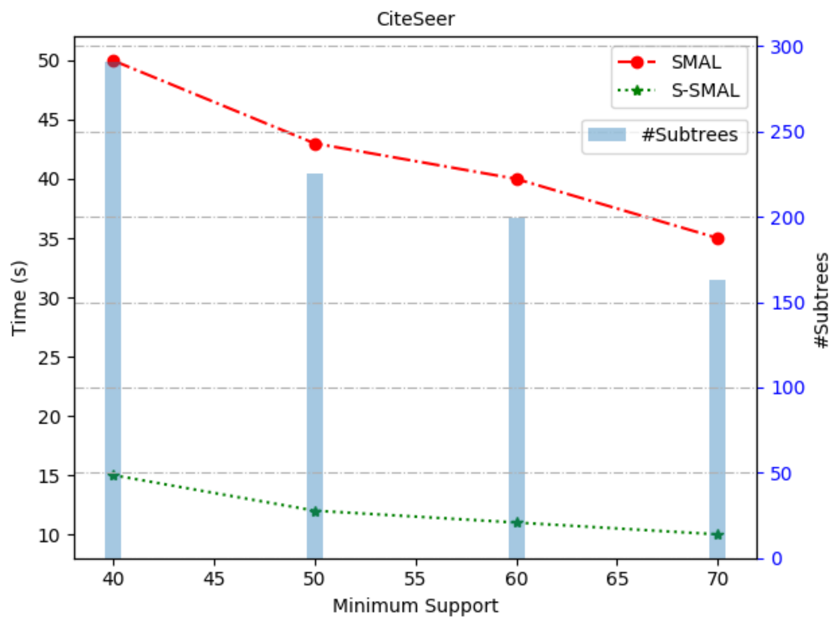


Figure 11: Results for Citeseer Graph

time in number of seconds. The blue bar represents the number of found frequent subtree patterns, where the right y-axis is the number of patterns. The performance of the algorithm degrades with decreasing minimum support because of generation of a large number of candidate subtrees. In both figures S-SMAL outperforms SMAL due to the distributed processing.

6.1. RUNTIMES DON'T SAY EVERYTHING

In recent years several frequent subtree mining algorithms were proposed. The authors of these new algorithms typically compared the runtimes of their implementations with those of previous implementations to confirm the efficiency of their methods. The runtime experiments however do not show whether the differences are due to a 'better'candidate generation algorithm or a 'better'candidate evaluation algorithm, as claimed by several authors, or due to the number of frequent subtrees resulting from the runs of all these algorithms [25].

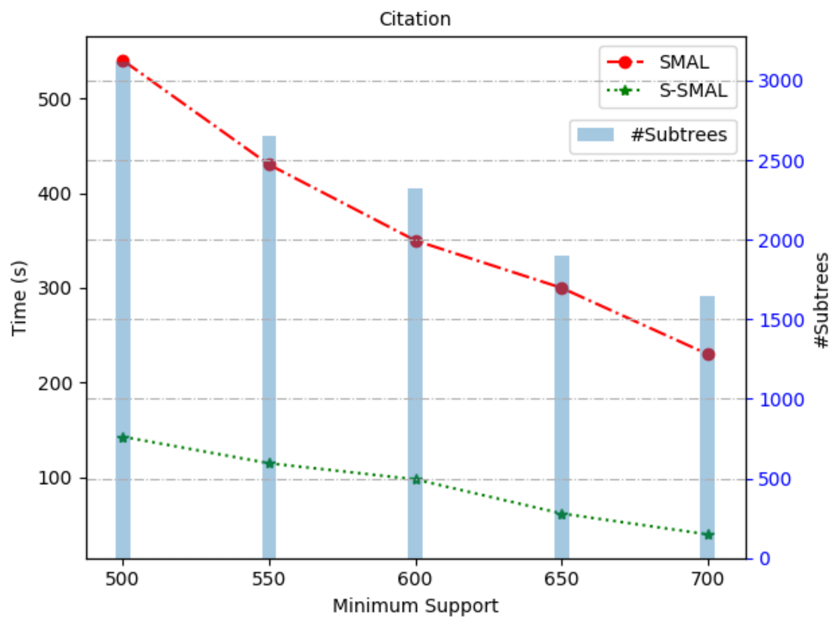


Figure 12: Results for Citation Graph

In an iterative algorithm candidate subtrees of size (k+1) are generated from size-k frequent subtrees. Hence in each iteration, if there are some false negatives (frequent subtrees that are not discovered), this can trigger more false negatives during next iterations that cumulatively can increase. Therefore, the false negatives can lead to missing valuable information. Additionally, comparing only the run-time of the algorithms without comparing the number of resulting frequent subtrees is unfair and insufficient to evaluate the performance of algorithms.

In this section we compare SMAL and HybridTreeMiner [24] in terms of the number of resulting frequent patterns using different values of minimum support. HybridTreeMiner is a well known frequent subtree mining algorithm that is designed based on breadth-first traversal and operates only on transaction datasets. When a frequent subtree mining algorithm such as SMAL is designed to operate on a single-graph setting, it can easily be changed to be applicable in a transactional setting. However, it is not true the other way around. One of the differences between these two settings is in regard to support count-

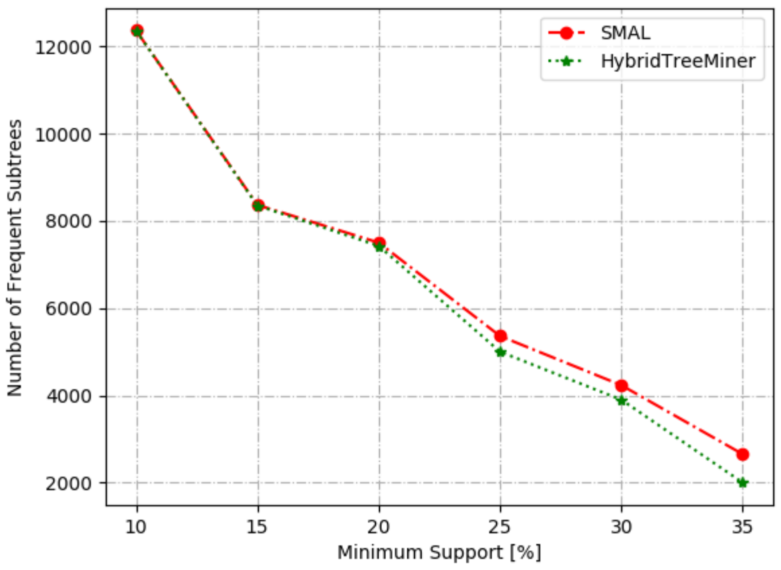


Figure 13: Comparing the number of frequent subtrees

ing for a candidate subtree. In a transactional setting the support of a subtree is defined by the number of graph transactions in which that subtree occurs, one count per transaction regardless of whether the subtree occurs once or more than once in a particular graph transaction. Whereas, in a single-graph setting the support of a subtree is defined by the number of occurrences of that subtree in the graph. On that note, we changed the support counting procedure of SMAL to be applicable on transactional settings. On that note, for the experiments we used a chemical compound dataset [22] which is a transaction dataset. The result is presented in [Figure 13] that compares the number of resulting frequent subtrees by both algorithms using different minimum support. In this dataset, there are 1000 transaction graphs where each graph has 50 vertices. As seen, SMAL can find and return more frequent subtrees, especially with higher value of minimum support. Depending on the interconnections of vertices in a graph and the value of minimum support, the false negatives may or may not happen while using an algorithm such as HyperTreeMiner. However, preventive procedures in SMAL avoid false negatives.

7. CONCLUSIONS AND FUTURE WORK

Herein we have presented SMAL, a frequent subtree mining algorithm which, unlike other algorithms, identifies, not simply the patterns, but all instances of the frequent subtrees found. This algorithm is efficiently implemented; it significantly avoids expensive isomorphism tests, and is designed for distributed computing. Experimental results on two real-world datasets show that a distributed implementation of SMAL outperforms its sequential implementation. Future work may be to implement SMAL using parallelization, rather than distributed computing, to see if even better efficiency can be achieved.

REFERENCES

[1] Babai, Laszlo. "Graph isomorphism in quasipolynomial time." Proceedings of the forty-eighth annual ACM symposium on Theory of Computing. ACM, 2016.

[2] Zhao, Peixiang, and Jeffrey Xu Yu. "Fast frequent free tree mining in graph databases." World Wide Web 11.1 (2008): 71-92.

[3] Telavekar, Dhananjay G., and Hemant A. Tirmare. "Frequent Pattern Mining On Unrooted Unordered Tree Using FRESTM." (2016).

[4] Kuramochi, Michihiro, and George Karypis. "Finding frequent patterns in a large sparse graph." Data mining and knowledge discovery 11.3 (2005): 243-271.

[5] Jiang, Chuntao, Frans Coenen, and Michele Zito. "A survey of frequent subgraph mining algorithms." The Knowledge Engineering Review 28.1 (2013): 75-105.

[6] Chi, Yun, Yirong Yang, and Richard R. Muntz. "Indexing and mining free trees." Data Mining, 2003. ICDM 2003. Third IEEE International Conference on. IEEE, 2003.

[7] Ruckert, Ulrich, and Stefan Kramer. "Frequent free tree discovery in graph data." Proceedings of the 2004 ACM symposium on Applied computing. ACM, 2004.

[8] Elseidy, Mohammed, et al. "Grami: Frequent subgraph and pattern mining in a single large graph." Proceedings of the VLDB Endowment 7.7 (2014): 517-528.

[9] Xin, Reynold S., et al. "Graphx: A resilient distributed graph system on spark." First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013.

[10] Shvachko, Konstantin, et al. "The hadoop distributed file system." Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on. Ieee, 2010.

[11] Vo, Bay, Dang Nguyen, and Thanh-Long Nguyen. "A parallel algorithm for frequent subgraph mining." Advanced Computational Methods for Knowledge Engineering. Springer, Cham, 2015. 163-173.

[12] Alonso, Andres Gago, et al. "Mining frequent connected subgraphs reducing the number of candidates." Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Berlin, Heidelberg, 2008.

[13] Dhiman, Aarzoo, and S. K. Jain. "Optimizing frequent subgraph mining for single large graph." Procedia Computer Science 89 (2016): 378-385.

[14] Zhao, Xiang, et al. "Frequent Subgraph Mining Based on Pregel." The Computer Journal 59.8 (2016): 1113-1128.

[15] Zou, Lei, et al. "PrefixTreeESpan: A pattern growth algorithm for mining embedded subtrees." International Conference on Web Information Systems Engineering. Springer, Berlin, Heidelberg, 2006.

[16] Han, Kun, et al. "Constrained Frequent Subtree Mining Method." Digital Home (ICDH), 2014 5th International Conference on. IEEE, 2014.

[17] Chi, Yun, Yirong Yang, and Richard R. Muntz. "Canonical forms for labelled trees and their applications in frequent subtree mining." Knowledge and Information Systems 8.2 (2005): 203-234.

[18] Zhao, Peixiang, and Jeffrey Xu Yu. "Mining closed frequent free trees in graph databases." International Conference on Database Systems for Advanced Applications. Springer, Berlin, Heidelberg, 2007.

[19] Abedijaberi, A., Eloe, N., Leopold, J., & Maryville, M. O. Interactive Visualization of Robustness Enhancement in Scale-free Networks with Limited Edge Addition (RENEA). Proceedings of the 23rd International Conference on Distributed Multimedia Systems, 2017.

[20] Prvzulj, N., Derek G. Corneil, and Igor Jurisica. "Efficient estimation of graphlet frequency distributions in protein-protein interaction networks." Bioinformatics 22.8 (2006): 974-980.

[21] Abedijaberi, Armita, and Jennifer Leopold. "FSMS: A Frequent Subgraph Mining Algorithm Using Mapping Sets." Machine Learning and Data Mining in Pattern Recognition. Springer, Cham, 2016. 761-773.

[22] Wang, Yanli, et al. "PubChem's BioAssay database." Nucleic acids research 40.D1 (2011): D400-D412.

[23] Yan, Xifeng, and Jiawei Han. "gspan: Graph-based substructure pattern mining." Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on. IEEE, 2002.

[24] Chi, Yun, Yirong Yang, and Richard R. Muntz. "HybridTreeMiner: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms." Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on. IEEE, 2004.

[25] Nijssen, Siegfried, and Joost Kok. "Frequent subgraph miners: runtimes don't say everything." Proceedings of the Workshop on Mining and Learning with Graphs. 2006.

III. INTERACTIVE VISUALIZATION OF ROBUSTNESS ENHANCEMENT IN SCALE-FREE NETWORKS WITH LIMITED EDGE ADDITION (RENEA)

Armita Abedijaberi, Nathan Eloe, and Jennifer Leopold

Department of Computer Science

Missouri University of Science and Technology, Rolla, Missouri 65401

Error tolerance and attack vulnerability of scale-free networks are usually used to evaluate the robustness of these networks. While new forms of attacks are developed everyday to compromise infrastructures, service providers are expected to develop strategies to mitigate the risk of extreme failures. Recently, much work has been devoted to design networks with optimal robustness, whereas little attention has been paid to improve the robustness of existing ones.

Herein we present RENEA, a method to improve the robustness of a scale-free network by adding a limited number of edges to it. While adding an edge to a network is an expensive task, our system, during each iteration, allows the user to select the best option based on the cost, amongst all proposed ones. The edge-addition interactions are performed through a visual user interface while the algorithm is running. RENEA is designed based on the evolution of the network's largest component during a sequence of targeted attacks. Through experiments on synthetic and real-life data sets, we conclude that applying RENEA on a scale-free network while interacting with the user, can drastically enforce its attack survivability at the lowest cost.

1. INTRODUCTION

One of the most important features of large networks is their degree distribuion, $P(k)$, or the probability that an arbitrary node be connected to exactly k other nodes. Many real-life networks display a power-law degree distribution with heavy-tailed statistics, which are called scale-free networks.

In a scale-free network the probability that a node has k links follows $P(k) \sim k^{(-\lambda)}$, where λ is called the degree exponent and its value is typically in the range between $2 < \lambda < 3$ [1]. Scale-free networks are created by preferential attachment [2], which means newly introduced nodes prefer to connect to existing high-degree nodes. Starting with a small number of nodes, when a new node is added to the network, considering the preferential linking, it will connect to other nodes with the probability proportional to their degree. Coupled with the expanding nature of many networks this explains the occurrence of hubs, that hold a much higher number of links than most of the nodes in the network. Scale-free topology is widely observed in many communication and transportation systems, such as the Internet, World Wide Web, airline networks, wireless sensor networks, power supply networks and etc.; all of which are essential to the modern society.

One of the most important properties in scale-free networks is the fact that while these networks are strongly tolerant against random failures, they are fragile under intentional attacks on the hubs. Intuition tells us that disabling a substantial number of nodes/edges will result in an inevitable functional disintegration of a network by breaking the network into tiny, non-communicating islands of nodes. However, scale-free networks can be amazingly resilient against accidental failures; even if 80% of randomly selected nodes fail, the remaining 20% still form a compact cluster with a path connecting any two nodes [3].

In fact, the fragileness of the scale-free networks under the intentional attack comes from their heavy-tailed property, causing loss of a large number of links when a hub node is crashed. Hence, the heavy loss of network links quickly makes the network sparsely connected and subsequently fragmented. However, random failures affect mainly the numerous small degree nodes, the absence of which doesn't disrupt the network's integrity. Considering error tolerance and attack vulnerability, which are two common and important properties of scale-free networks, extensive research efforts have been made to study the robustness of such networks which is defined as the ability of the surviving nodes to remain, as much as possible, interconnected.

On that account it is important to understand how to design networks which are optimally robust against malicious attacks, with examples of terrorist attacks on physical networks or attacks by hackers on computer networks. However, it is not possible to abandon the existing networks, which are the result of years of evolution, and rebuild them from the beginning.

Hence, it is significantly important to study the optimizing guidelines to enhance the robustness of existing networks in an interactive environment to incorporate the user's input in order to acquire the optimal result at a lower cost.

In this paper, we study the problem of how to improve the robustness of an existing scale-free network and show that its attack survivability can be enforced greatly by adding a limited number of edges to it at the lowest cost. This process is carried on via visualizing the process and interaction with the user, whilst having no impact on the error tolerance through keeping the degree distribution of the network as much as possible intact.

The organization of this paper is as follows; Section II provides an overview of related work about robustness enhancement in scale-free networks. In the section III we discuss RENEA and its graphical user interface. Section IV focuses on experiments and results. An example of running RENEA is presented in Section V. Finally, conclusions are presented in Section VI.

2. RELATED WORK

The main approaches to improve robustness of scale-free networks, through topology reconfiguration, can be generally classified into two main categories. The first category involves rewiring edges of the network and the second category suggests addition of edges to the network.

Both approaches are carried out in order to obtain a network structure with better robustness. Herein we summarize existing work review regarding these two approaches.

2.1. RECONFIGURATION WITH EDGE REWIRING

Four different schemes are proposed to increase the robustness of a network represented as a graph via edge rewiring [4]. In these methods edges are selected randomly or based on specific criteria to get removed from the graph or to be added to the graph. According to results, preferential edge selection outperforms random edge selection. However, this reconfiguration will change the degree distribution of the graph as well as its diameter; which is not desirable for the network.

Another edge rewiring method is proposed in [5] to enhance the robustness of the scale-free network without changing any nodal degree. This method repetitively applies the rewiring operation in the original network by dividing nodes with degrees strictly lower than T into two non-overlapping parts, with degrees less than and greater than T_1, respectively. The degree of a certain node i is denoted as k_i. Two edges $A-B$ and $C-D$ with this condition $k_A > T > k_B > T_1 \gg max(k_C, k_D)$, are selected. If $A-C$ and $B-D$ do not already exist in the network, the rewiring operation replaces $A-B$ and $C-D$ with $A-C$ and $B-D$ as long as such reconfiguration does not generate a loop. This rewiring operation obviously does not change any nodal degree since it converts a random network into another one with the same degree distribution. However, for real networks with economic constraints, the nodal degree conservation is not enough due to the cost.

In fact, the total length of links cannot be exceedingly large and the number of changes in the network should remain small. To deal with this concern, the authors in [6], [7] use another edge swapping method while having a fewer number of swapped edges during the reconfiguration process. This method is similar to the method used in [5] except that edges $A-B$ and $C-D$ are selected without fulfilling the aforementioned conditions. In order to minimize the cost, any swap is accepted, only if the increase of the robustness is greater than or equal to a threshold. This procedure is repeated with another randomly chosen pair of edges until no further substantial improvement is achieved. This method results in a network with an onion-like structure [6], [7]. Even though swapping the edges can increase the network robustness to some extent, there are some spatially limited real-life networks where edges are hard to re-configure, e.g. power grid networks and the Internet router network. On the other hand, however, there exist the spatial unlimited networks such as airline networks and the Internet switcher network.

For spatially unlimited networks whose edges can be easily re-linked, in the Switch Link (SL) method [9], the top P_c fraction of the large-degree nodes are defined as hubs. For each hub, the *SL* finds two non-hub nodes connecting it. Keeps the edge connecting to first non-hub node and switches the edge connecting the second non-hub node to the first non-hub node. This process will be repeated until all the links connected to the hub nodes are addressed. For the spatially limited networks, the SL method is not economic and feasible since nodes are usually far from each other. In this case, split hub (SH) method is proposed [9]. This method also starts with defining the top P_c fraction of nodes as hubs and replaces them by a 3-clique which is a complete graph in which every two distinct nodes are adjacent. Then it connects the non-hub nodes, that were connected to the original hub node, to the nodes of the clique randomly.

Edge reconfiguration can change community structure of a network [10]. The community structure refers to the functional modules in the network that play an important role in regards to cascading failures. A Network with a strong community structure has a few number of edges between its communities. Hence, its structure is more fragile in terms of attacks on those edges in comparison with networks with a weak community structure. A method [10] is proposed to improve the robustness of a network while preserving its community structure. In this 3-step method, the importance of nodes is calculated based on their degree. Step 1 reconfigures each community to have an onion-like structure as explained in [6] and [7]. Step 2 swaps edges in such a way that nodes with high importance only connect to the nodes within the same community. Step 3 swaps edges to increase the number of edges between communities. These 3 steps are recursively applied on the network until its robustness is hard to increase.

2.2. RECONFIGURATION WITH EDGE ADDITION

The number of possibible ways of adding an edge to a graph with N nodes and L edges is equal to $\binom{N}{2} - L$. For large real-life (sparse) networks, it is almost infeasible to compare all these possibilities and find the optimal edges to add. Some techniques have been introduced in literature to add edges to an existing graph based on different criteria. Three different enforcing strategies are practiced in [11]. The first method randomly selects a pair of nodes in the network and establishes a new edge between them. The second method selects a pair of nodes with the lowest degree in the network and establishes a new edge between them. Finally, the last method selects a pair of nodes with the highest degree in the network and establishes a new edge between them. According to experiments, the method that prefers low-degree nodes as candidates for adding edges, can reinforces the attack survivability of the network with a lower cost in comparison with the other methods.

It has been suggested [12] that assortative networks (i.e., high-degree nodes in the network are more likely linked with other high-degree nodes), are more robust than their disassortative counterparts (i.e., high-degree nodes in the network are more likely linked with low-degree nodes). Thereupon, a method is proposed [12] to enhance robustness of a network by increasing its assortativity. In this method, a layer index is assigned to each node based on the degree of that node. Accordingly, the layer index for nodes with the lowest degree is 0, for nodes with the second lowest degree is 1, and so on. Then, the probability of adding an edge between a random pair of nodes depends on their layer index difference. Hence, nodes within the same layer are connected with greater probabilities than nodes in different layers; which leads to higher assortativity in the network. The way that nodes are arranged in a graph is considered as a key factor in overall robustness and efficiency of that graph. The star structure is efficient (the average shortest path length is small), yet fragile in case of removal of the central node. On the contrary, the circle structure is robust with regard to removal of any single node, yet inefficient (the average shortest path length is large). The Node-protecting Cycle methode [13] is proposed to combine the properties of both circle and star structures to improve robustness and efficiency of a network.

All of the aforementioned edge rewiring and edge addition algorithms are proposed to improve the robustness of an existing network. However, these works consider one aspect in a network and neglect the other. When it comes to a scale-free network, it is important to take every structural aspect of the network into consideration. In addition, feasibility of a proposed re-configuration of a network must be given thought to. Having a method to improve the robustness of a scale-free network against malicious attacks while keeping its resilience in case of random failures without disturbing its small-world property at a very low cost is still an unsolved problem.

2.3. NETWORK VISUALIZATION

A number of software tools have been developed to model, analyze, and visualize data that can be represented as a graph or a network. Typically, these tools allow the user to annotate nodes and edges with metadata, and provide utilities such as random graph generation, calculation of analytical measures (e.g., centrality, network distances, PageRank, etc.), and filtering (i.e., viewing only a portion of the graph based on some criteria). Several of these viewers were designed for a particular problem domain, such as finding motifs in gene regulatory networks (e.g., GeneNetWeaver [14]). KDnuggets provides a list of what it considers to be the top 30 network visualization tools that can be used for a wide variety of network applications (e.g., in domains such as biology, finance, and sociology). It is notable that many of these tools are open-source and can be customized to provide additional functionality. The software presented herein differs from other available network visualization tools in the analysis that it facilitates. To the best of our knowledge, there is no other visual network analysis tools available for the purpose of reinforcing robustness of a scale-free network in an interactive environment.

3. ROBUSTNESS ENHANCEMENT ALGORITHM

The problem setting we consider in this work is to modify a given graph's structure under a given budget to improve its robustness. The budget which is often in terms of maximum number of edges that can be added to the network. The measure of robustness employed in our algorithm and the edge-addition strategy along with its graphical user interface are specified in the following sections.

3.1. ROBUSTNESS METRIC

The definition of network robustness might change according to a specific application. In this work, the removal of a node from a network is called a "node-knockout" or a "node-attack", and the robustness of a network is measured by the size of the Largest Connected Component (LCC) in the network after a node-attack [7]. To quantify it, we proceed with a series of node-attacks and subsequently measure the robustness after each node-removal. Hence, the robustness R is defined as:

$$R = \frac{1}{N} \sum_{q=1/N}^{1} S(q) \tag{1}$$

where N is the number of nodes in the network and $S(q)$ is the fraction of nodes in the LCC after removing qN nodes. The normalization factor $1/N$, makes robustness of networks with different sizes comparable. The minimum of R is equal to $1/N$, where the network is a star graph and the maximum of R is equal to 0.5, where the network is a complete graph. A robust network is generally corresponding to a large R value. This distinctive measure is not only simple, but also practical, due to the calculation of the size of the largest component during all possible system-wide failures or intentional attacks [10]. In our targeted attack scenario, we implicitly admit that the attacker perfectly knows the network's degree sequence and thus can cause maximum damage. Since an intentional attack is targeted to disrupt the network by removing the most important nodes; here, we find the most connected node, remove it along with all the edges emanating from it, calculate $S(q)$, update the degree sequence for the remaining nodes, and find the new most connected node to repeat the process until the network completely collapses. In case of two or more nodes having the same degree, we simply pick one of them randomly.

3.2. RENEA

Here we present an algorithm called RENEA to improve the robustness of a scale-free network by adding a limited number of edges to it, such that the optimized network, compared to the initial one, has significantly higher value of robustness. We assume that the 'defender' knows about the intention of the 'attacker' to cause maximum damage to the network and thus acts upon through our Algorithm.

For scale-free graphs there is always a giant component which is a connected component of the graph that contains a constant fraction of the entire graph's nodes. Resultantly, one can randomly remove more than 80% of the nodes in the graph without destroying the giant component. Hence, the network will still possess a large-scale connectivity [8]. On the other hand, an attack that simultaneously eliminates as few as 10–20% of hubs can cause the giant component to disappear suddenly and therefore break the network into several isolated components. The main idea in our method is to increase the robustness of a network by adding a limited number of edges to it, and therefore to elongate the lifetime of the giant component of the graph upon removing high-degree nodes.

Depends upon the nature of a network, adding an edge can be very costly and some networks can only afford a few of them. Moreover, in order to keep other structural properties of such networks, essentially their degree distribution as much intact as possible, it requires addition of a limited number of edges to the network. Given that, a threshold parameter is needed for the algorithm to confine the maximum number of edges that can be added to the network. Furthermore, our iterative algorithm provides the user with a number of edge-additional candidates during each iteration and let them choose the one that cause the minimum cost in order to be added the network.

The Inputs to our iterative algorithm RENEA are an undirected, unweighted scale-free network represented as a graph $G(V,E)$ with $|V| = N$ nodes and $|E| = L$ edges, the budget δ (maximum number of edges that can be added to G), and the initial percentage θ for removing hubs in the attack simulation process. The Output from RENEA is a more robust version of graph $G(V,E')$ with the same number of nodes yet more number of edges ($L \leq |E'| \leq L+\delta$). The steps of the algorithm are as follows:

```
0) $m = 0$   // m is the number of edges added to G
1) Simulate a targeted attack and remove $\theta$ percent of hubs from $G$.
2) $listComps$ = Sorted list of disconnected components based on their size, decreasingly.
3) $comp1$ = The largest component in $listComps$.
4) $comp2$ = The second largest component in $listComps$.
5) Find a node $x \in comp1$
6) Find a node $y \in comp2$
7) $E' = \{x\text{–}y\} \cup E$.   // add x–y to G
8) $comp = comp1 + comp2$
9) Remove $comp1$ and $comp2$ from $listComps$.
10) Add $comp$ to the beginning of $listComps$.
11) $m = m + 1$
12) Repeat steps 3-11 until $|listComps| == 1$ or $m == \delta$.
13) If $m < \delta$ then $\theta = \theta + \tau$ and goto step 1.
```

RENEA starts off with simulating an attack on graph G (line 1). There are two common types of attacks that can be carried out on a network known as serial-attack and sudden-attack. In both of these attacks an initial θ percent of highest-degree nodes are to be removed.

In the sudden-attack, θ percent of highest-degree hubs are appointed and removed at once, whereas in the serial-attack, which is known to be more damaging, hub removal happens a bit differently. In the serial-attack, first the highest-degree hub is removed, then the degree of remaining nodes are recalculated and among them again the highest-degree hub is removed until θ percent of hubs are discarded. In the graphical user interface, a user can choose either of these options to simulate an attack on the network.

Once G is fragmented, all disconnected components are found, where the number of nodes in each component determines the size of that component. In order to add the least number of edges yet gain the highest improvement in robustness, the single-node components will be ignored. $listComps$ contains a list of components sorted based on their size in a descending order (line 2). The first and second members of $listComps$, the largest and second largest components, are called $comp1$ and $comp2$, respectively (lines 3-4). Add an edge between node x in $comp1$ and node y in $comp2$. There exist a list of candidates whose size is equal to $|comp1| \times |comp2|$.

Our user interface allows the user to pick one that can impose the least cost to the network (lines 5-7). Afterwards, $comp1$ and $comp2$ are combined into $comp$ and both are removed from $listComps$, while $comp$ is added to the beginning of $listComps$ (to keep it sorted) (lines 8-10). Variable m that is used to track the number of edges added to G is updated (line 11). At this point one edge is already added to G ($E' = E \cup \{x\text{–}y\}$) and m=1.

RENEA, as an iterative algorithm, repeats steps 3-11 until $listComps$ contains only one component (at the end of each iteration, size of $listComps$ decreases by one) or the desired number of edges are added to G ($|E'| = L + \delta$) (line 12). Ultimately, if the size of $listCompos$ reaches to one yet still $m < \delta$, the value of θ can increase by τ and go to step 1.

τ is a small number that can determined by the user. For our experiments, heuristically, we set the value of θ to 20 and τ to 5.

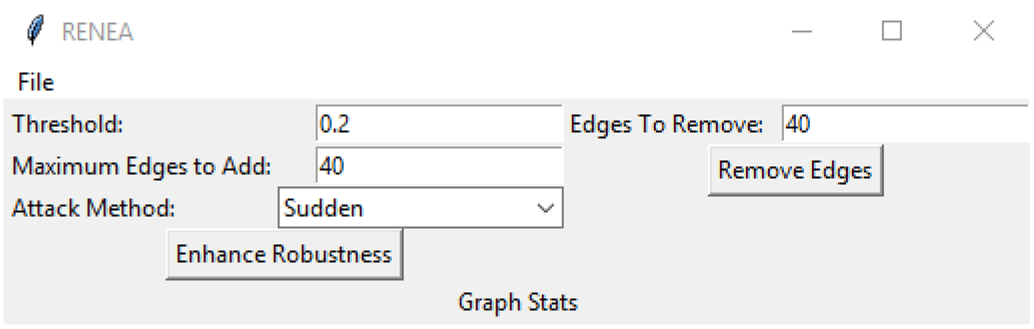
3.3. EDGE REMOVAL

Even though applying RENEA on a graph improves its robustness remarkably, adding more edges to a network comes with an extra cost. Hence, depends on the nature of the network, some users may or may not decide to mitigate the total cost by getting rid of some edges from the graph yet still have a considerable overall enhancement in the robustness of the network. Upon request of the user, our algorithm nominates some edges to get removed from the graph based on the betweeness centrality value of them. The edge betweeness centrality is defined as the number of the shortest paths that pass through an edge of a network. An edge with a high edge betweeness centrality score represents a bridge-like connector between two parts of a network and the removal of which may affect the communication between many pairs of nodes through the shortest path between them. To implement the edge-removal part, first off each edge is associated with an edge centrality value. Then, these values are sorted in an increasing order.

The user can request the maximum number of edges that is willing to remove from the network. Removing edges with high betweeness, that occupy critical roles in the network, can force many pairs of nodes to re-route on a longer way in order to communicate with each other. It can also degrades the overall efficiency of the network in terms of communication. Hence, the algorithm starts removing edges with the lowest betweeness value, as long as that edge-removal does not make the network disconnected, until desired number of edges are removed.

3.4. THE GRAPHICAL USER INTERFACE

RENEA is implemented using the Python programming language using the powerful networkx library to perform graph operations. The user interface uses elements of the


RENEA

File

Threshold: 0.2

Edges To Remove: 40

Maximum Edges to Add: 40

Remove Edges

Attack Method: Sudden

Enhance Robustness

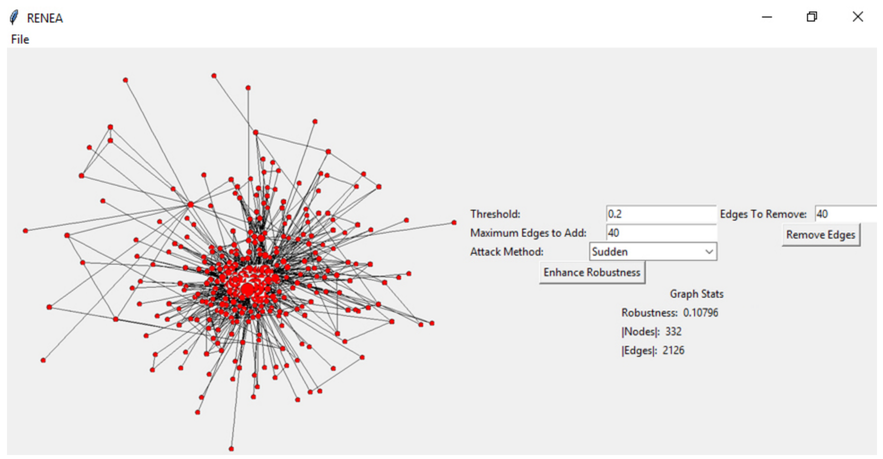
Graph Stats

Figure 1: RENEA GUI

networx_viewer to easily enable an interactive view of the graph that allows scrolling, zooming, and moving nodes. The Graph Viewer element is included in a Tkinter based UI; Tkinter is the standard UI library in Python and is included in a variety of distributions of the language. This helps to ensure that the application is as cross platform as possible.

The Graphical User Interface (GUI) of RENEA [Figure 1] consists of the following controls: (i) a file chooser to allow the user to select a file that contains a list of nodes and edges of the graph, (ii) a text input field to specify the initial value of θ, (iii) a text input field to specify the maximum number of edges to be added to the graph, (vi) a drop-down menu to select the desired type of attack on the graph, (vi) a control button to start process of adding edges to the graph, (v) a text input field to specify the maximum number of edges to get removed from the graph, and (vi) a control button to start the process of removing edges from the graph.

As shown in [Figure 2], the usAir data set is uploaded, and all of its specifications are presented in the GUI; such as the number of nodes, edges, and initial robustness. The important nodes in terms of their degree (hubs) are presented using bigger circles. In this GUI user can drag and relocate each edge and node and zoom in/out on the graph for a closer look.



Threshold: 0.2
Edges To Remove: 40
Maximum Edges to Add: 40
Attack Method: Sudden

Graph Stats
Robustness: 0.10796
|Nodes|: 332
|Edges|: 2126

Figure 2: RENEA GUI after uploading the US Air graph, all the specifications of this graph are presented in the GUI.

4. EXPERIMENTS

In this section, we experimentally examine the performance of RENEA, and for performance evaluation comparison, we have also implemented three other edge-addition algorithms. One to add edges between randomly selected pairs of nodes, one to add edges between nodes with high degrees only, and finally one to add edges between low-degree nodes. For the experiment, we used a real-life American Airlines connection dataset which is an unweighted, undirected, and connected scale-free graph. It also contains no multi-edges (two or more edges that are incident to the same two nodes) or self-loops (an edge from a node to itself).

4.1. RESULTS

We tested RENEA and three other algorithms on American Airline network. We ran each algorithm 10 times and took the average of improvement acquired by each. For each experiment, we added 20, 30, 40, 50, and 60 edges to the original graph using different algorithms and computed the obained robustness. As shown in [*Table 1*], RENEA out-

Table 1: The performance of RENEA vs. three other algorithms Algo1, Algo2, and Algo3. Algo1 adds edges randomly, Algo2 adds edges among low-degree nodes, and Algo3 adds edges among high-degree nodes. The data set is American Airline dataset with 332 nodes, 2126 edges and initial robustness of 0.1079. We added 20, 30, 40, 50, and 60 edges to the original graph and the averages of total robustness improvement acquired after running each algorithm are presented.

R:0.1079	20 e	30 e	40 e	50 e	60 e
RENEA	26.17%	38.16%	49.94%	55.25%	66.09%
Algo1	11.08%	18.19%	23.20%	30.44%	34.07%
Algo2	11.55%	13.17%	15.42%	21.80%	24.60%
Algo3	10.26%	12.14%	19.09%	21.51%	21.72%

performs the other algorithms by far and accomplishes two times more improvements in comparison with the other ones. Results show that by adding less than 3% of edges to the graph, its robustness improves up to 70%, whereas the random edge addition can only improve the robustness up to 35%.

[Figure 3] is also representing how the size of biggest component in the American Airline network is changing while removing q hubs from the original network. The size of largest component after applying RENEA, causes the graph to hold its integrity for a longer time as opposed to using other methods.

[Figure 4] compares the degree distribution of the US Air network before and after adding 50 edges via applying RENEA. Because of adding a limited number of edges to the graph, the shape of its degree distribution remains almost the same. The reason that a scale-free network is robust in terms of random failure is because of having a power-low degree distribution and adding a limited number of edges to the graph does not cause a severe changes in it. We agenda is to keep all the unique properties of the graph the same while enhancing its robustness against targeted attacks.

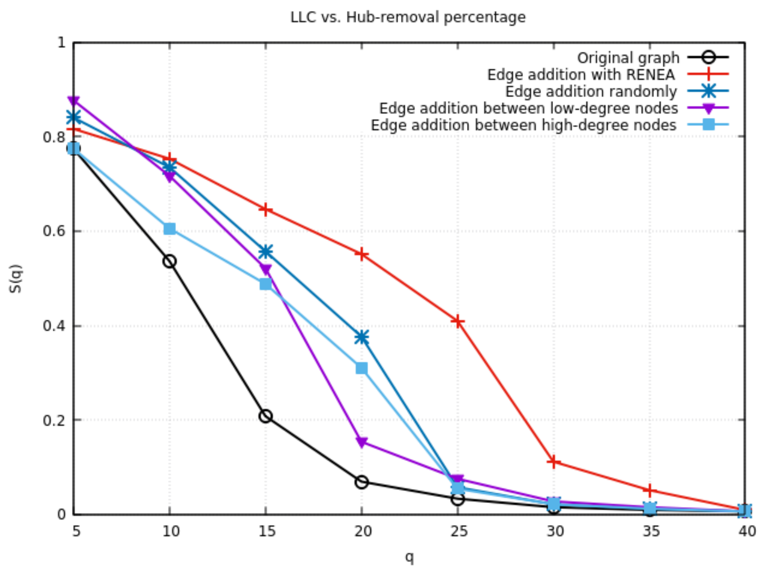


Figure 3: x-axis represents the number of removed nodes and y-axis represents the size of LLC in the American Airline network before and after reconfiguration. Line in red shows the performance of RENEA.

5. A RUNNING EXAMPLE IN RENEA

To demonstrate the concepts behind RENEA, here we walk through a simple example. We start by uploading a graph to the GUI with 20 nodes, 21 edges, and the initial robustness of 0.1052. Once the graph is uploaded, all of its specifications are presented on the GUI [Figure 5]. For this particular graph, we set *'Threshold' = 20%*, *'Maximum Edges to Add' = 3*, and *'Attack Method' = Serial*. Once the user hits the *'Enhance Robustness'* button, the algorithm runs and suggests a list of edge IDs to be added. User can accept or discard the proposed edges [Figure 6]. Once the user accepts these edges they will be added to graph (they are represeted with thicker lines in blue) [Figure 7]. If the user chooses to delete 3 edges, once they hit the *'Remove Edges'* button, the algorithm calculates the betweeness of the edges and nominates maximum three edges with the lowest betweeness whose removal does not disconnect the graph. These edges are colored in red with thicker

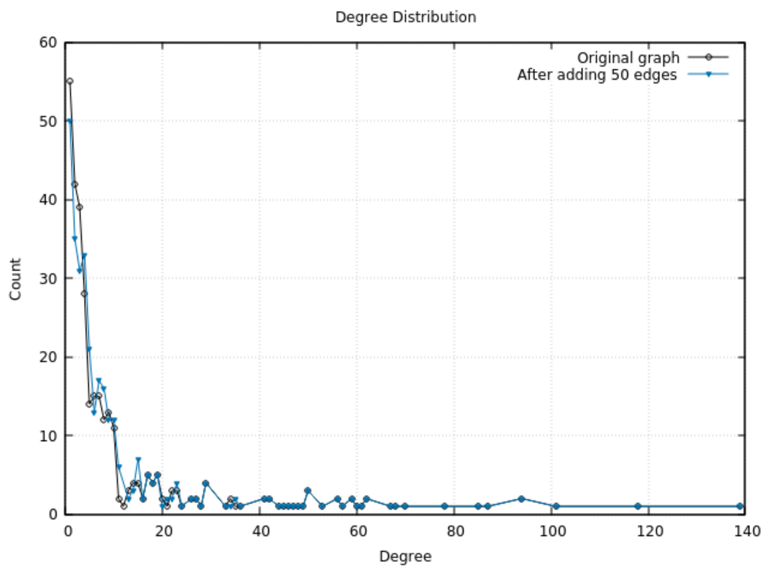
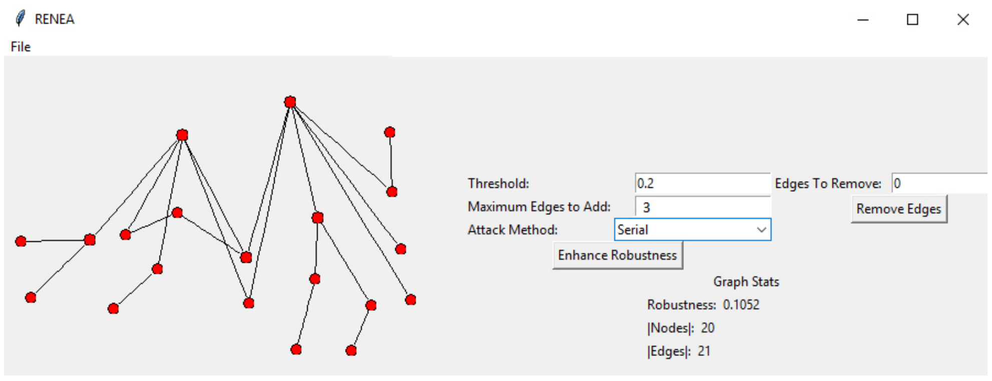


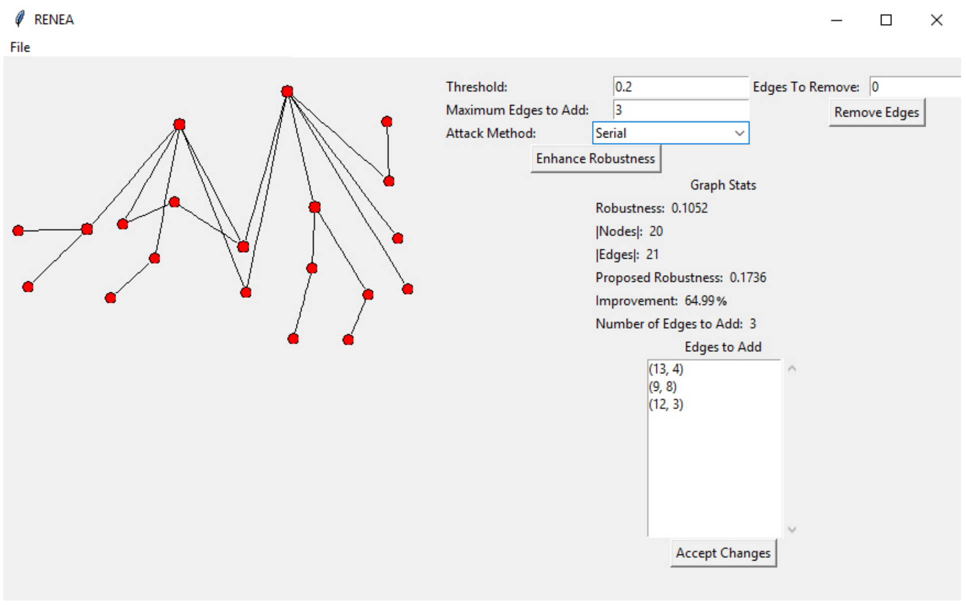
Figure 4: x-axis represents degrees of nodes and y-axis represents how many nodes share the same degree. After adding 50 edges to US Air network, its degree distribution remains almost the same.



Threshold: 0.2
Edges To Remove: 0
Maximum Edges to Add: 3
Attack Method: Serial
Robustness: 0.1052
|Nodes|: 20
|Edges|: 21

Figure 5: A graph with 20 nodes and 21 edges is uploaded.

lines [Figure 8]. The GUI also shows the amount of decrease in robustness due to edge removal. Once the user accepts the changes those edges get removed and final robustness along with the final graph is shown [Figure 9].

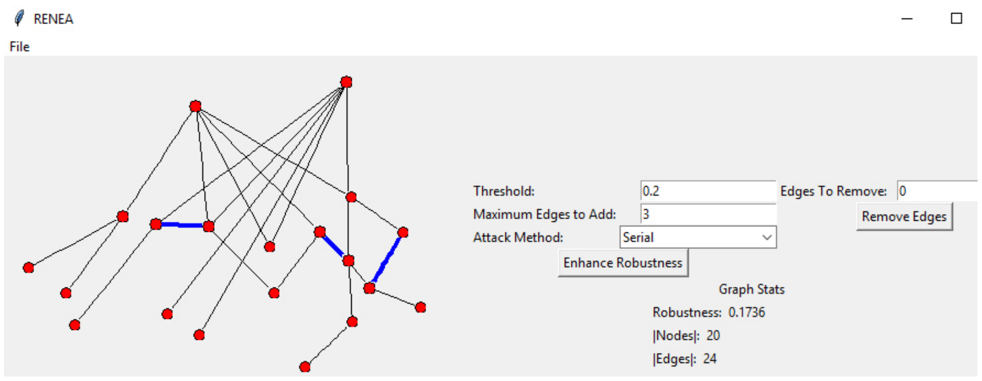


Threshold: 0.2
Edges To Remove: 0
Maximum Edges to Add: 3
Attack Method: Serial

Robustness: 0.1052
|Nodes|: 20
|Edges|: 21
Proposed Robustness: 0.1736
Improvement: 64.99 %
Number of Edges to Add: 3

(13, 4)
(9, 8)
(12, 3)

Figure 6: A list of suggested edges to add in order to enhance its robustness.



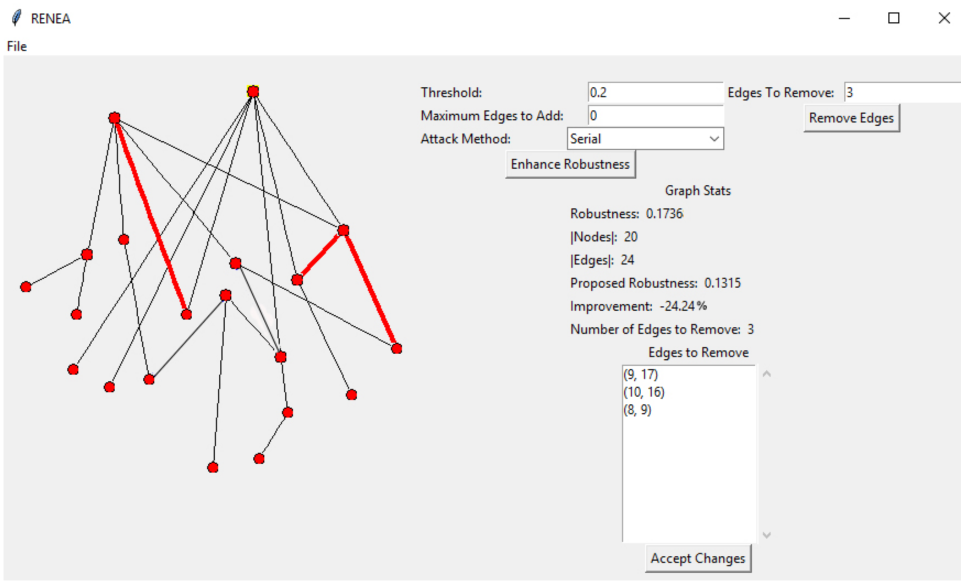
Threshold: 0.2
Edges To Remove: 0
Maximum Edges to Add: 3
Attack Method: Serial

Robustness: 0.1736
|Nodes|: 20
|Edges|: 24

Figure 7: Graph after adding 3 edges.

6. CONCLUSION

Real-life complex networks are known to be resilient in terms of random failures yet fragile in the terms of targeted attacks. To enhance their total robustness, one strategy is to add more edges to the network and the other is to rewire some qualified edges in the

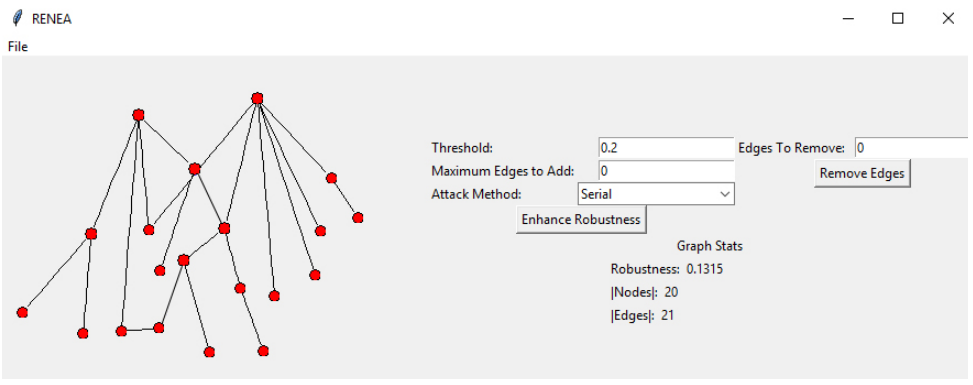


Threshold: 0.2
Edges To Remove: 3
Maximum Edges to Add: 0
Attack Method: Serial

Graph Stats
Robustness: 0.1736
|Nodes|: 20
|Edges|: 24
Proposed Robustness: 0.1315
Improvement: -24.24%
Number of Edges to Remove: 3

Edges to Remove
(9, 17)
(10, 16)
(8, 9)

Figure 8: 3 edges are nominated to get removed.



Threshold: 0.2
Edges To Remove: 0
Maximum Edges to Add: 0
Attack Method: Serial

Graph Stats
Robustness: 0.1315
|Nodes|: 20
|Edges|: 21

Figure 9: After adding 3 edges and removing 3 edges, the overall robustness improvement is 25%.

network. Based on the type of the network either of these methods could be used. Mostly these strategies accept a change only if it improves the robustness of the network by a threshold. Hence, it requires for them to compute the robustness of the graph before and after applying a change which could be very time-consuming depends on the size of the graph. So, they are not considered being efficient in terms of time. Additionally, none of these solutions take the nature of the network into consideration. Thus, their proposed solution is not always a practical one. In this work, we presented RENEA, an iterative algorithm that is designed to enhance the overall robustness of a scale-free network against malicious attacks by adding a limited number of edges to it. Adding new connections to the nodes arbitrarily and without any constraint can change the nodal degree of the graph and disturbs other structural properties of it. We also presented a user interface for RENEA that interacts with the user. In addition to the excellent performance of the RENEA, the fact that RENEA does not compute the robustness during each iteration, makes it work very fast to be able to communicate with the user. During each iteration, RENEA suggests the best edges that adding them can increase the robustness of a graph the most, and the user, considering the cost, can accept or reject them. Finally, if the user also wants to remove some edges from the graph to mitigate the overall cost, RENEA nominates the ones with low betweeness centrality value whose removal does not make the graph disconnected.

REFERENCES

[1] Cohen, R., and Havlin, S. (2003). Scale-free networks are ultrasmall. Physical review letters, 90(5), 05

[2] Barabasi, A. L., and Albert, R. (1999). Emergence of scaling in random networks. science, 286(5439), 509-512. Chicago

[3] Barabasi, A. L., and Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization. Nature reviews genetics, 5(2), 101-113.

[4] Beygelzimer, A., Grinstein, G., Linsker, R., and Rish, I. (2005). Improving network robustness by edge modification. Physica A: Statistical Mechanics and its Applications, 357(3), 593-612.

[5] Xiao, S., Xiao, G., Cheng, T. H., Ma, S., Fu, X., and Soh, H. (2010). Robustness of scale-free networks under rewiring operations. EPL (Europhysics Letters), 89(3), 38002.

[6] Schneider, C. M., Moreira, A., Andrade, J. S., Havlin, S., and Herrmann, H. J. (2010). Onion-like network topology enhances robustness against malicious attacks. J. Stat. Mech.

[7] Schneider, C. M., Moreira, A. A., Andrade, J. S., Havlin, S., and Herrmann, H. J. (2011). Mitigation of malicious attacks on networks. Proceedings of the National Academy of Sciences, 108(10), 3838-3841.

[8] Callaway, D. S., Newman, M. E., Strogatz, S. H., and Watts, D. J. (2000). Network robustness and fragility: Percolation on random graphs. Physical review letters, 85(25), 5468.

[9] Ze-Hui, Q., Pu, W., Chao-Ming, S., and Zhi-Guang, Q. (2010). Enhancement of scale-free network attack tolerance. Chinese Physics B, 19(11), 110504.

[10]] Yang, Y., Li, Z., Chen, Y., Zhang, X., and Wang, S. (2015). Improving the Robustness of Complex Networks with Preserving Community Structure. PloS one, 10(2), e0116551.

[11] Zhao, J., and Xu, K. (2009). Enhancing the robustness of scale-free networks. Journal of Physics A: Mathematical and Theoretical, 42(19), 195003.

[12] Wu, Z. X., and Holme, P. (2011). Onion structure and network robustness. Physical Review E, 84(2), 026106.

[13] Li, L., Jia, Q. S., Guan, X., and Wang, H. (2012). Enhancing the robustness and efficiency of scale-free network with limited link addition. KSII Transactions on Internet and Information Systems (TIIS), 6(5), 1333-1353.

[14] Schaffter, Thomas, Daniel Marbach, and Dario Floreano. "GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods." Bioinformatics 27.16 (2011): 2263-2270.

IV. INTERACTIVE VISUALIZATION OF ROBUSTNESS ENHANCEMENT IN SCALE-FREE NETWORKS AGAINST CASCADING FAILURES

Armita Abedijaberi, Jennifer Leopold, and Nathan Eloe

Department of Computer Science

Missouri University of Science and Technology, Rolla, Missouri 65401

Error tolerance and attack vulnerability are two common and important properties of scale-free networks, which are usually used to evaluate the robustness of these networks. While new forms of attacks are developed everyday to compromise infrastructures, service providers are also expected to develop strategies to mitigate the risk of extreme failures. On that note, cascading failures are crucial issues in the study of survivability of infrastructures and have attracted a lot of research interest. Recently, much work has been devoted to the design of networks with optimal robustness. However, little attention has been paid to improving the robustness of existing networks. Herein we present a method to improve the robustness of a scale-free network by adding a limited number of edges to it. We then assign adjustable weights to each individual edge of the network. This approach is based on the evolution of the largest connected component of the network after a targeted attack on the high-degree vertices. Another novel aspect of our work is that the edge-addition interactions are performed through a visual interface while the algorithm is running. Through extensive experiments, using both synthetic and real-world networks, we conclude that applying our method on a scale-free network, can drastically enforce the attack survivability of a network.

1. INTRODUCTION

One of the important features of large networks is their degree distribution, $P(k)$, or the probability that an arbitrary vertex is connected to k other vertices. Many real-life networks display a power-law degree distribution with heavy-tailed statistics, which are called scale-free networks. In a scale-free network the probability that a vertex has k links (edges) follows $P(k) \sim k^{(-\lambda)}$, where λ is the degree exponent and its value is typically in the range between $2 < \lambda < 3$ [2].

Scale-free networks are created by preferential attachment [8], which means when a new vertex is added to the network, it will connect to other vertices with a probability proportional to their degree. Because of the preferential attachment, a large majority of vertices have low degree, but a small number, known as "hubs", have high degree. Scale-free topology is widely observed in many communication and transportation systems, such as the Internet, World Wide Web, airline networks, wireless sensor networks, power supply networks, etc., all of which are essential to modern society.

One of the most important properties in scale-free networks is that while these networks are strongly tolerant against random failures, they are fragile under intentional attacks on the hubs. Intuition tells us that disabling a substantial number of vertices and edges will result in an inevitable functional disintegration of the network by breaking it into small, non-communicating islands of vertices. However, scale-free networks can be amazingly resilient against accidental failures; even if 80% of randomly selected vertices fail, the remaining 20% still form a compact cluster with a path connecting any pair of vertices [9]. In fact, the fragileness of scale-free networks under the intentional attack emanates from their heavy-tailed property, causing loss of a large number of links when a hub is disabled. Hence, the heavy loss of network links quickly makes the network sparsely connected and subsequently fragmented. However, random failures mainly affect the small-degree vertices, the absence of which doesn't disrupt the network integrity.

Considering error tolerance and attack vulnerability, which are two common and important properties of scale-free networks, extensive research efforts have been made to study the robustness of such networks; this is defined as the ability of the surviving vertices to remain, as much as possible, interconnected. Hence, it is important to understand how to design networks that are optimally robust against malicious attacks, with examples of terrorist attacks on physical networks or attacks by hackers on computer networks. However, it is not possible to abandon the existing networks, which are the result of years of evolution, and rebuild them from the beginning. Therefore, it is significantly essential to study the optimizing guidelines to enhance the robustness of existing networks in an interactive environment to incorporate the user's input in order to acquire the optimal result at a lower cost.

In this paper, we study the problem of how to improve the robustness of a scale-free network. We show that the attack survivability of a network can be greatly enhanced by adding a limited number of edges to it and assigning adjustable weights to the edges. This procedure is carried out via visualizing the process and interacting with the user. The organization of this paper is as follows. Section 2 provides an overview of related work about robustness enhancement in scale-free networks. Section 3 introduces the model of cascading failures in scale-free networks induced by attacks of highest-degree vertices. In Section 4 we discuss our proposed algorithm and its graphical user interface. Section 5 focuses on experiments and results. Finally, conclusions are presented in Section 6.

2. RELATED WORK

The networked systems are routinely subjected to external shocks, where the breakdown of one or more heavily loaded vertices will cause the redistribution of loads over the remaining vertices. As a result, this can trigger breakdowns of newly overloaded vertices. These additional failures require a new redistribution of loads, that either stabilizes if failures get locally absorbed, or grows until a large number of vertices are compromised to a failure point [8]. There are two groups of existing methods that are designed to improve the robustness of a scale-free network against cascading failures. One group of methods enhances the network robustness *statically*. In this approach, the goal is to prevent the cascading failures from happening before the occurrence of initial failures in the network. For instance, in [1], the topological structure of an input graph is altered to improve its overall robustness. The trade-off in such graph evolvement is the cost and changes in structural properties of the graph such as its community structures. The other group of methods is designed to enhance the network robustness *dynamically*. In a dynamic approach the goal is to handle the cascading failures and minimize the damages after the initial failures have occurred. For instance, a relatively costless defense method [10] is proposed based on selectively removing some vertices and edges immediately following an initial attack and before the propagation of the cascade. Experiments shows that this intentional removal of network elements can drastically reduce the size of the cascade. However, it requires the early detection of cascading failures.

Alternatively, the strategies to mitigate the cascading failures are divided into two categories called *hard* strategies and *soft* strategies. The hard strategies attempt to minimize the damages caused by cascading failures by changing the connections in the network. Whereas the soft strategies attempt to minimize the damages caused by cascading failures without changing the connections in the network. The hard strategies can be classified further into two main categories. The first category involves rewiring edges of the network and the second category suggests addition of edges to the network. Both approaches are carried out in order to obtain a network structure with better robustness. We summarize a review of existing work regarding these two approaches in this section.

2.1. RECONFIGURATION WITH EDGE REWIRING

Four different schemes have been proposed to increase the robustness of a network, represented as a graph, via edge rewiring [16]. In these methods edges are selected randomly or based on specific criteria to be removed from the graph or to be added to the graph. According to experimental results, preferential edge selection outperforms random edge selection. However, this reconfiguration will change the degree distribution and diameter of the network.

Another edge rewiring method is proposed [17] to enhance the robustness of a scale-free network without changing its degree distribution. This method, repeatedly, selects two edges $A-B$ and $C-D$ from the network. If $A-C$ and $B-D$ do not already exist in the network, the rewiring operation replaces $A-B$ and $C-D$ with $A-C$ and $B-D$, as long as such reconfiguration does not create a loop. Obviously, this rewiring operation does not change any nodal degree since it converts one network into another network with the same degree distribution. However, for real-life networks with economic constraints, the nodal degree conservation is not enough. For instance, the total geographic length of the edges should not increase and the number of changes in the network should remain small. To deal with this concern, the authors in [18], [27] use another edge swapping method to decrease the number of swapped edges during the reconfiguration process. This method is similar to the one in [17] except that herein any swap is accepted as long as the increase in the value of robustness is greater than or equal to a predefined threshold. This procedure repeats until no further substantial improvement is achieved for a large number of consecutive swapping trials. This method results in a network with an onion-like structure [18], [27].

Even though swapping the edges can increase the network robustness to some extent, there are some spatially limited real-life networks where it is hard to reconfigure edges, e.g. power grid networks and the Internet router network. On the other hand, however, there also exist spatially unlimited networks such as airline networks and the Internet switcher network. For spatially unlimited networks where edges can be easily re-linked, the Switch Link (SL) method [20] is proposed. For each hub vertex of the network, SL finds two non-hub vertices that are connected to the hub vertex. Then, it keeps the edge between the hub vertex and the first non-hub vertex, and disconnects the second non-hub vertex from the hub vertex and connects it to the first non-hub vertex. This process will repeat until all the links connected to the hub vertices are addressed or desired robustness improvement is achieved. For the spatially limited networks, the SL method is not economic and feasible since vertices are usually located far from each other. In this case, the split hub (SH) method is proposed [20]. This method starts with replacing each hub vertex by a 3-clique (a complete graph in which every two distinct vertices are adjacent). Then, it connects the non-hub vertices, that were connected to the original hub vertex, to the vertices of the 3-clique, randomly.

Edge reconfiguration can also change the community structure of a network [21]. The community structure refers to the occurrence of groups of vertices in a network that are more densely connected internally than with the rest of the network. A network with a strong community structure has only a small number of edges between its communities. Hence, its structure is more fragile in terms of attacks on those edges in comparison with networks with a weak community structure. A method in [21] is proposed to improve the robustness of a network while preserving its community structure. In this 3-step method, the importance of vertices is calculated based on their degree. Step 1 reconfigures each community to have an onion-like structure as explained in [18] and [27]. Step 2 swaps edges in such a way that vertices with high importance only connect to vertices within the same community. Step 3 swaps edges to increase the number of edges between communities. These three steps are recursively applied on the network until no further substantial robustness improvement is achieved for a large number of consecutive swapping trials.

2.2. RECONFIGURATION WITH EDGE ADDITION

The number of possible ways of adding an edge to a graph with N vertices and L edges is equal to $\binom{N}{2} - L$. For large real-life (sparse) networks, it is almost infeasible to compare all these possibilities and find the optimal edges to add. Three different enforcing strategies are discussed in [23] to add edges to a network to increase its robustness. The first method randomly selects a pair of vertices in the network and establishes a new edge between them. The second method selects a pair of vertices with the lowest degree in the network and establishes a new edge between them. Finally, the last method selects a pair of vertices with the highest degree in the network and establishes a new edge between them. According to experiments, the method that prefers low-degree vertices as candidates for adding edges outperforms the other two methods in terms of attack survivability reinforcement of the network.

It has been suggested [24] that assortative networks (i.e., high-degree vertices in the network are more likely linked with other high-degree vertices), are more robust than their disassortative counterparts (i.e., high-degree vertices in the network are more likely linked with low-degree vertices). Thereupon, a method is proposed in [24] to enhance robustness of a network by increasing its assortativity. In this method, a layer index is assigned to each vertex based on the degree of that vertex. Accordingly, the layer index for vertices with the lowest degree is 0, for vertices with the second lowest degree is 1, and so on. The probability of adding an edge between a random pair of vertices depends on their layer index difference. Hence, vertices within the same layer are connected with greater probabilities than vertices in different layers, which leads to higher assortativity in the network.

The way that vertices are arranged in a graph is considered as a key factor in the overall robustness and efficiency of the graph. The shortest path between two vertices u and v is the path with the smallest length joining u to v. The star structure is efficient (the average shortest path length is small because all the communication between any pair of vertices passes through the central vertex), yet fragile in case of removal of the central vertex. In contrast, the circle structure is robust with regard to removal of any single vertex, yet inefficient (the average shortest path length is large). The Vertex-protecting Cycle method in [25] is proposed to combine the properties of both circle and star structures to improve robustness and efficiency of a network.

All of the aforementioned edge-addition algorithms are proposed to improve the robustness of an existing network. However, these works consider one aspect in a network and neglect the other properties. When it comes to a scale-free network, it is important to take every structural aspect of the network into consideration. In addition, feasibility of reconfiguration of a network must be considered. Having a method to improve the robustness of a scale-free network against malicious attacks while retaining its efficiency [section 4.2] is still an unsolved problem.

2.3. NETWORK VISUALIZATION

A number of software tools have been developed to model, analyze, and visualize data that can be represented as a network (graph). Typically, these tools allow the user to annotate vertices and edges with metadata, and provide utilities such as random graph generation, calculation of analytical measures (e.g., centrality, network distances, PageRank), and filtering (i.e., viewing only a portion of the graph based on some criteria). Several of these viewers were designed for a particular problem domain, such as finding motifs in gene regulatory networks (e.g., GeneNetWeaver [12]). KDnuggets provides a list of what it considers to be the top 30 network visualization tools that can be used for a wide variety of network applications (e.g., biology, finance, sociology). It is notable that many of these tools are open-source and can be customized to provide additional functionality. To the best of our knowledge, there are no other network visualization tools available for the purpose of reinforcing robustness of a scale-free network in an interactive environment.

3. MODELING THE CASCADING FAILURES IN SCALE-FREE NETWORKS

In this section, the cascading failure in a scale-free network is modeled under a vertex-based attack. For a given network, physical flows (such as electric currents, information, etc.) are exchanged between pairs of vertices. A cascading failure in the network happens when the failure of a vertex triggers the failure of successive vertices. When one part of a system fails, nearby vertices must then compensate for the failed component. This in turn can overload these vertices, causing them to fail as well, with additional vertices failing one after another. In addition to vertex overload, invalid connectivity is another reason that leads to cascading failures. Invalid connectivity happens if a certain number of vertices, that are still present in the network, lose their paths to the largest connected component after failure of high-degree vertices.

Usually, for any pair of vertices, the physical flows are exchanged and transmitted along the shortest paths connecting them [5]. The shortest path, or the most efficient path, is defined as the minimum weighted path length between two vertices. The weight w_{ij} of each edge e_{ij} initially is set to one for all the edges, indicating an equivalent transport of flow between vertices. The length of the shortest path between vertex i and vertex j if they are directly connected is $l_{ij} = w_{ij} = 1$ and if they are not directly connected, but are connected through intermediate vertices, then:

$$l_{ij} = w_{ik_1} + w_{k_1k_2} + ... + w_{k_tj}, k_t \in K \tag{1}$$

where K is the set of intermediate vertices belonging to the shortest path between vertex i and vertex j. Initially, at time $t = 0$, weight values of all edges are set to one.

The transmitting capacity of a vertex can be measured by its *load*, which is equal to the amount of flows that the vertex is requested to transmit. This depends on the total number of shortest weighted paths passing through the vertex. Therefore, the load $L_i(t)$ of vertex i at time t is measured by the betweenness centrality of vertex i, denoted as follows:

$$L_i(t) = \sum_{i \neq j \neq k \in V} \frac{\sigma_{jk}(i)}{\sigma_{jk}} \tag{2}$$

where V is the set of vertices in the network G, σ_{jk} is the total number of shortest paths between vertex j and k and $\sigma_{jk}(i)$ is the total number of shortest paths between vertex j and k that pass through vertex i. The maximum load that a vertex can handle is defined as its capacity. The capacity C_i of vertex i is assumed to be proportional to its initial load $L_i(0)$:

$$C_i = (1+\alpha)L_i(0), \forall i \tag{3}$$

where $L_i(0)$ represents the initial load before the attack and $\alpha \geq 0$ is the tolerance parameter of network G that captures the relationship between the vertex capacity and load demand levels. Here, α also implies the budget of network construction or resource allocation [8]. The larger α is, the higher capacity a vertex i has, hence the higher resilience against failures. However, due to the cost of the construction of components (vertices or links), the capacity should be finitely large [7]. Initially ($t = 0$), the network is assumed to be in a stationary state, which means all the vertices are enabled with loads lower than their capacities. The evolving procedure of cascading failures begins with the removal of some vertices in the network. Consequently, the load on other vertices will change and be redistributed over the entire network. At some time t, the vertex i will fail if the new load $L_i(t)$ on vertex i exceeds its capacity C_i. This will cause the new redistribution of loads over the network. This process is iterated until there is no vertex exceeding its capacity. At this time, the iterative process can be regarded as being completed. This iterative process is called *cascading failures* [6]. A common, yet hard-to-predict, property of cascading failures is that the damages caused by a single point of failure could propagate widely and result in a global collapse. Two basic models of cascading failures include *percolation cascade* and *capacity cascade* [11]. In percolation cascades, the state of vertices changes due to influence of their neighbors. Whereas in the capacity cascade, the failure of a vertex can propagate and affect vertices far away from it. Herein we focus on capacity cascade failure.

4. ROBUSTNESS ENHANCEMENT ALGORITHM

The problem setting we consider in this work is to modify a network structure under a given budget in order to improve its robustness. The budget is specified as the maximum number of edges that can be added to the network. The measure of robustness employed in our algorithm, the edge-addition strategy, the weight-assignment procedure, and the graphical user interface are discussed in the following subsections.

4.1. ROBUSTNESS METRIC

The definition of network robustness might change according to a specific application. In this work, the removal of a vertex from a network is called a *vertex-knockout* or a *vertex-attack*, and the robustness R of a network is quantified in terms of the relative size of the largest connected component (LCC) as follows:

$$R = \frac{N'}{N} \tag{4}$$

where N and N' are the numbers of vertices in the largest connected component before and after the cascading failures, respectively. N is the size of the initial network and $0 \leq R \leq 1$.

$R \approx 1$ generally corresponds to a robust network, i.e. there is no cascade in the network and all vertices are connected and functional (i.e., enabled) after the initial attack. On the other hand, $R \approx 0$ indicates that the network has been disconnected into several small sub-networks after the initial attack.

This distinctive measure is also aligned with the *robust-yet-fragile* property of scale-free networks, i.e. R remains close to unity in the case of random breakdowns, but is significantly reduced under attacks that target hubs.

In our targeted attack scenario, we implicitly admit that the attacker perfectly knows the network degree sequence and thus can cause maximum damage. Since an intentional attack is targeted to disrupt the network by removing the most important vertices, here we consider the vertices in descending order of initial degree and remove the vertex with the highest degree along with all the edges emanating from it. We then calculate R, update the load for the remaining vertices, and remove the ones with capacities smaller than their loads. We repeat this process until the network completely collapses. In case of two or more vertices having the same degree, we simply pick one of them randomly.

4.2. EFFICIENCY

The shortest path between vertices in a network represents the ability of the network to transmit flow from one vertex to other vertices. The average value of the shortest path between all vertices reflects the general ability of the whole system. Therefore, the system efficiency is used to evaluate the system performance, which can be obtained from the efficiency of the shortest path between vertex i and vertex j. The efficiency of a network G is:

$$E(G) = \frac{1}{N(N-1)} \sum_{i \neq j \in G} (\varepsilon_{ij}) \tag{5}$$

where N is the number of vertices in G and ε_{ij} is the reciprocal of the length of the shortest path between a vertex i and another vertex j.

4.3. RENEA

Here we present a Robustness Enhancement algorithm in scale-free Networks with Limited Edge Addition (RENEA) where each individual edge has a designated weight, such that the optimized network, compared to the initial one, has a significantly higher value of robustness. We assume that the *defender* knows about the intention of the *attacker* to cause maximum damage to the network and thus takes preventative actions using our algorithm. Scale-free graphs always have a giant component which is a connected component of the graph that contains a constant fraction of vertices. Resultantly, one can randomly remove more than 80% of the vertices in the graph without destroying the giant component, while the network will still possess a large-scale connectivity [4]. On the other hand, an attack that simultaneously eliminates as few as 10–20% of the hubs can cause the giant component to disappear suddenly and therefore break the network into several isolated components. Hubs are vertices in the network with a high number of connections and usually serve as common connections to mediate the shortest path length between vertices. Hence, if hubs get attacked (disabled), the network will become fragmented into disconnected components with invalid connections between them. When a vertex is disabled, all the edges connecting to it will be disabled too (invalid connections). Thus, we first want to create alternative paths between disconnected components to maintain the connectivity of the network [25].

The main idea in our method is to increase the robustness of a network by 1) adding a limited number of edges to it in order to eliminate invalid connectivity, and 2) assigning a weight to each edge, thereby mitigating cascading failures upon removing high-degree vertices (hubs). Depending upon the type of the network, adding an edge can be very costly. For example, adding edges (Fiber optics) between vertices (continents) in the optical network will cost too much [21]. Given that, a threshold parameter is needed for the algorithm to limit the maximum number of edges that can be added to the network. Furthermore, our iterative algorithm provides the user with a list of candidate edges during each iteration and lets him/her choose among them. The inputs to our iterative algorithm RENEA are an undirected scale-free network represented as an attributed graph $G(V,E,L_V,C_V)$ with $|V| = N$ vertices and $|E| = L$ edges, and the budget δ (maximum number of edges that can be added to G). L_V is a list that contains the initial load of each vertex, C_V is a list that contains the capacity of each vertex, and R is the robustness value of G. We assume that there are no multi-edges (i.e., two or more edges that are incident to the same two vertices) or self-loops (i.e., an edge from a vertex to itself) in the input graph, and the weight of all the edges is one.

The output from RENEA is the graph $G(V,E',L'_V,C'_V)$ with the same number of vertices yet more edges ($L \leq |E'| \leq L+\delta$). R' is the robustness value of G ($R' > R$). The load and capacity of each of the vertices are also updated in the output graph. The steps of the algorithm are as follows.

Algorithm 2: Edge Addition

```
Input: Graph $G(V,E,L_V,C_V)$ and budget $\delta$        // $w_{i,j}=1\ \forall i,j \in V$
Output: Graph $G(V,E',L'_V,C'_V)$        // $|E| \leq |E'| \leq |E|+\delta$
1: $m = 0$
2: $E' \leftarrow E$
3: $excessLoad \leftarrow \emptyset$
4: A targeted attack is simulated and hubs are removed from $G(V,E,L_V,C_V)$
5: $listComps \leftarrow$ Sorted list of disconnected components based on their size
6: repeat
7:   $comp1 \leftarrow$ The largest component in $listComps$.
8:   $comp2 \leftarrow$ The second largest component in $listComps$.
9:   for each vertex $i$ in $comp1$ do
10:    for each vertex $j$ in $comp2$ do
11:      $E' = e_{ij} \cup E'$
12:      $excessLoad[e_{ij}] = 0$
13:      $L'_V \leftarrow$ Recompute the load of each vertex
14:      for each vertex $v$ in $V$ do
15:        if $L'_v > L_v$ then
16:          $excessLoad[e_{ij}] = excessLoad[e_{ij}] + |L'_v - L_v|$
17:        end if
18:      end for
19:      $E' = E' \backslash e_{ij}$
20:    end for
21:  end for
22:  $e_{ij} \leftarrow$ The edge with the smallest value in excessLoad table
23:  $E' = e_{ij} \cup E'$
24:  $L_V \leftarrow$ Recompute the load of each vertex
25:  $comp = comp1 + comp2$
26:  Remove $comp1$ and $comp2$ from $listComps$.
27:  Add $comp$ to the beginning of $listComps$
28:  $m = m + 1$
29: until $|listComps| == 1$ or $m == \delta$
30: $L'_V \leftarrow$ Recompute the load of each vertex
31: $C'_V \leftarrow$ Recompute the load of each vertex
32: return $G(V,E',L'_V,C'_V)$
```

RENEA starts with simulating an attack on the hubs of the input graph $G(V,E,L_V,C_V)$ (line 4). There are two common types of attacks that can be carried out on a network known as *serial-attack* and *sudden-attack*. In both of these attacks, hubs are targeted to be disabled. In the sudden-attack, all hubs are selected and removed at once. In the serial-attack that is known to be more damaging, first the highest-degree vertex is removed, then the degree of remaining vertices are recalculated and among them again the highest-degree vertex is removed until the network has collapsed and malfunctioned (i.e., the size of the largest connected component is nearly 10% of the initial network [29]). In the graphical user interface, a user can choose either of these options to simulate an attack on the network.

Once G has collapsed, all disconnected components are found, where the number of vertices in each component determines the size of that component. $listComps$ contains a list of components sorted based on their size in a non-increasing order (line 5). The first and second members of $listComps$, which are the largest and second largest components, are called $comp1$ and $comp2$, respectively (lines 7-8).

We add an edge between vertex i in $comp1$ and vertex j in $comp2$. There exists a list of candidates with a size equal to $|comp1| \times |comp2|$. We add each candidate edge to the graph and compute the total excess loads on each vertex caused by an additional edge and keep that value in a hash table called *excessLoad* (lines 9-21). Finally, the edge that causes the least amount of excess loads on vertices will be selected to be added to G (lines 22-23). However, our user interface allows the user to pick any of the candidate edges considering the type of network. Since adding an edge could change the loads of vertices in the graph, the loads for all the vertices are recomputed (line 24). Afterwards, $comp1$ and $comp2$ are combined into $comp$ and both are removed from $listComps$, while $comp$ is added to the beginning of $listComps$ (to keep it sorted) (lines 25-27). The variable m, that is used to track the number of edges added to G, is updated at the end of each iteration (line 28). After the first iteration, one edge is added to G and m=1. RENEA repeats steps 7-28 iteratively until $listComps$ contains only one component or the desired number of edges are added to G ($|E'| = L + \delta$) (line 29). Since the size of listComps is monotonically decreasing and the value of m is monotonically increasing, RENEA eventually terminates and returns G (line 32).

4.4. EDGE WEIGHT ASSIGNMENT

According to [4], certain topological properties of networks, such as their degree distribution, have strong impacts on their stability. In terms of the degree distribution (the probability distribution of the degrees of all the vertices in the network), complex networks can be classified into homogeneous and heterogeneous networks. Homogeneous networks such as random graphs [14] possess a binomial degree distribution, where the vertex degrees concentrate around the mean degree. Heterogeneous networks such as scale-free networks have a heavy-tailed degree distribution that follows a power law, where there is a huge variability between vertex degrees, with several orders of magnitude between them. A heterogeneous degree distribution in a scale-free network makes it resistant against random failures. However, targeted malicious attacks can easily disrupt the network by removing only a small fraction of vertices or links. On the other hand, a network with homogeneous degree distribution, could be more robust against attacks and random failures [5].

As mentioned earlier, the load of a vertex is defined based on its betweenness centrality which is strongly correlated with its degree [28]. Hence, scale-free networks with a heterogeneous degree distribution also have a heterogeneous load distribution [30]. This means that there is a small group of vertices which have large betweenness but the majority of vertices in the network have small betweenness. Thus, physical flows, that are transmitted between pairs of vertices, will have a high probability to pass through this small number of high-betweenness vertices.

The betweenness distribution in random networks is in general more homogeneous [31]. Consequently, in random networks the weighted shortest paths are less dependent on the heavily linked vertices. Thus, removal of such vertices is less probable to trigger cascading failures. That is a supporting factor for the explanation as to why random networks are more tolerant to cascading failures. On that account, to increase the homogeneity of the load distribution in a network we use a weighting method for edges of the network. This mitigation strategy aims at reducing the load on each of the vertices in the network. This reduction in load attempts to keep the vertices operating below their maximum capacities in order to better accommodate the redistribution of loads due to failure of high-load vertices.

We set the weight w_{ij} (flow) of an arbitrary edge, connecting vertices i and j, proportionally to the connectedness of two vertices as follows:

$$w_{ij} = (k_i k_j)^{\beta} \tag{6}$$

where β is a tunable weight parameter, governing the strength of the edge weight, and k_i and k_j are the degrees of vertices i and j, respectively. The intuition for this weighting method is that an edge is important in a network when its two end point vertices are important. So, the betweenness of the edge has positive correlation with the product of vertex degrees at both ends of it. The value of tuning parameter β can lead to different scenarios; $\beta > 0$ indicates that edges connecting hubs have high weights, and are avoided to be used to transmit the flow, $\beta = 0$ indicates that all edges have a weight equal to one, and $\beta < 0$ indicates that edges connecting hubs have low weights, and are frequently used to transmit the flow.

Weight assignment can directly affect the load distribution of vertices when the weighted shortest path is used to transmit the flow along the edges of the network. Additionally, tuning the value of β from -1 to $+1$ changes the load distribution of the network from heterogeneous to homogeneous by reducing the standard deviation of betweenness in the network. The standard deviation of betweenness is defined as

$$\delta_B = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(B_i - \langle B \rangle)^2} \tag{7}$$

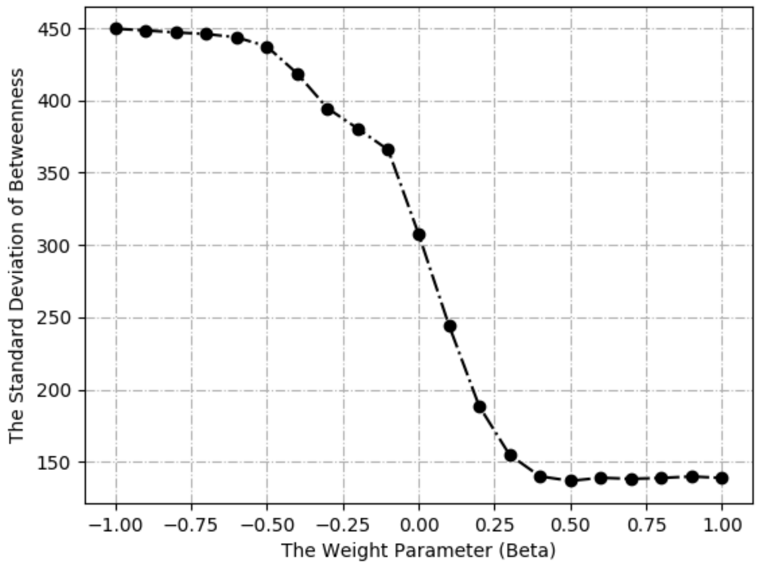


Figure 1: The standard deviation of betweenness vs. the Weight Parameter (Beta) for a scale-free network with 1000 vertices and 1567 edges with degree exponent of 2.5. Increasing the value of β from -1 to $+1$ results in decreasing the value of δ_B. This figure shows a transition from a heterogeneous load distribution to a homogeneous load distribution.

where $\langle B \rangle$ is the average betweenness of the network. When $\beta = -1$, the scale-free network shows a more heterogeneous load distribution (i.e., hubs are more frequently used to transmit information). Hence, there will be an increase in the value of δ_B in the network. When $\beta = +1$, the scale-free network shows a more homogeneous load distribution (i.e., hubs experience a significant decrease in load and vertices which used to carry a small load may acquire a higher load). Hence, there will be a decrease in the value of δ_B in the network. In a network with a homogeneous load distribution, all vertices contribute equally to transmit the flow along the network as shown in [Figure 1].

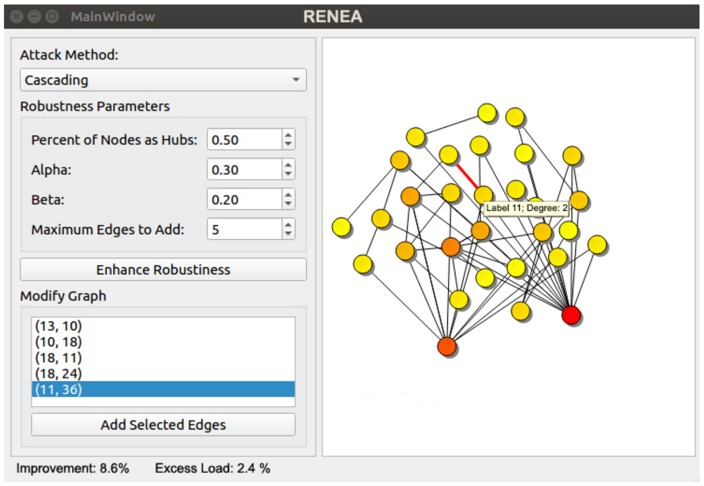
Selecting the appropriate value of β is crucial for the robustness of the network. If the value of β is set close to -1, the loads on some vertices are relatively large and their removal is likely to start a sequence of overload failures. Setting the value of β close to

$+1$ can decrease the efficiency of the network. Finding an intermediate value of β can gradually transform the network from having a heterogeneous load distribution to a more homogeneous one. In the experiment section we show how we assign weights to each edge of the network and tune the value of β in order to increase robustness yet still maintain the efficiency of the network.

4.5. THE GRAPHICAL USER INTERFACE

RENEA is implemented using the Python programming language using the powerful networkx library to perform graph operations. The graph viewer is a custom written QT Widget which provides for an interactive interface that allows for moving of individual vertices, inspection of vertices (via tooltips), and zooming in and out of the graph view. This widget is embedded in an interface written using QT bindings for Python. As all of these tools are supported on major operating systems (Linux, macOS, and Windows), this tool can be considered cross-platform. The Graphical User Interface (GUI) of RENEA [Figure 2] consists of the following controls: (i) a file chooser to allow the user to select a file that contains a list of vertices and edges of the graph, (ii) a text input field to specify the initial value of α, (iii) a text field to show the value of β (value of β can also be specified), (iv) a text input field to specify the maximum number of edges to be added to the graph, (v) a drop-down list to select the desired type of attack on the graph, (vi) a control button to start the process of adding edges to the graph and assigning weights to the edges, and (vii) a menu to let the user choose from the suggested edges.

As shown in [Figure 2], a data set has been uploaded. The important vertices in terms of their loads (hubs) are presented using darker colors. In this GUI the user can drag and relocate each edge and vertex and zoom in/out on the graph for a closer look. The user can also use the mouse to hover over a vertex to see its id. Vertex ids are not displayed all the time to avoid cluttering the display (e.g., an airline graph can contain hundreds of vertices). To see the corresponding edge in the graph display, the user can use



Attack Method: Cascading

Robustness Parameters

Percent of Nodes as Hubs: 0.50

Alpha: 0.30

Beta: 0.20

Maximum Edges to Add: 5

Modify Graph

(13, 10)
(10, 18)
(18, 11)
(18, 24)
(11, 36)

Improvement: 8.6%   Excess Load: 2.4 %

Figure 2: RENEA GUI

the mouse to hover over the edges in the graph; the vertex end points of the moused-over edge will be highlighted in the left panel display. For each suggested edge the value of robustness improvement and the value of excess load will be calculated and shown to the user.

5. EXPERIMENTS

In this section, we experimentally examine the performance of RENEA. For performance evaluation comparison, we also implemented four other algorithms that are designed to enhance the robustness of a network by adding edges amongst its vertices. In particular, for the two end points of a new connecting edge in Random Addition algorithm (RA) two vertices are randomly selected, in Low-degree Addition algorithm (LA) two vertices with the lowest degrees are selected, in Low Betweenness algorithm (LB) two vertices with the lowest betweenness are selected [23], and in Algebraic Connectivity Based algorithm (ACB) two vertices with the largest algebraic connectivity are selected [26]. In all of these algorithms, the process of adding edges will continue until the designated number of edges is reached.

We conducted simulations with both artificial and real-life networks. We used two scale-free networks generated by the Brabasi-Albert model [11] as benchmark networks (one scale-free network with 1000 vertices and 1567 edges, and the other scale-free network with 1500 vertices and 2016 edges where both have the degree exponent (λ) of 2.5). We also used a real-life American Airlines connection dataset (with 332 vertices and 2126 edges) and a network of routers in the Internet (with 2026 vertices and 4233 edges). In order to have more reliable statistics, each simulation was repeated 10 times and the average was shown in the results.

5.1. RESULTS

We first show the effect of adding edges to a network without assigning weights to the edges. As shown in [Figure 3], the size of the largest connected component after removal of hubs is larger when RENEA is applied on the networks in comparison with applying the other algorithms. We next show the effect of adding weights to the edges of a network and the relationship between the tolerance parameter α and the weight parameter β. Intuitively, the most effective method to enhance the robustness of a network is to prevent cascading failures by increasing the value of α so that all vertices have sufficient resources to prevent failure due to overload. However, the capacity is often limited by cost, thus it is impractical to assign excessively large capacity to all vertices of a network. As shown in [Figure 4], the robustness of a network can be enhanced by adjusting the values of α and β. The results show that adding weights to the edges can enhance the robustness of a network without increasing the capacity of vertices.

We added up to only 2% of the original number of edges to each graph. We also experimented with different values of $-1 \leq \beta \leq +1$ to find which value has the highest robustness improvement and lowest efficiency trade-off. To do so, starting with $\beta = 1$, we gradually increased the the value of β to $\beta = +1$ by a step value of 0.1. We then recorded the values of percentage increase in robustness and percentage decrease in efficiency of

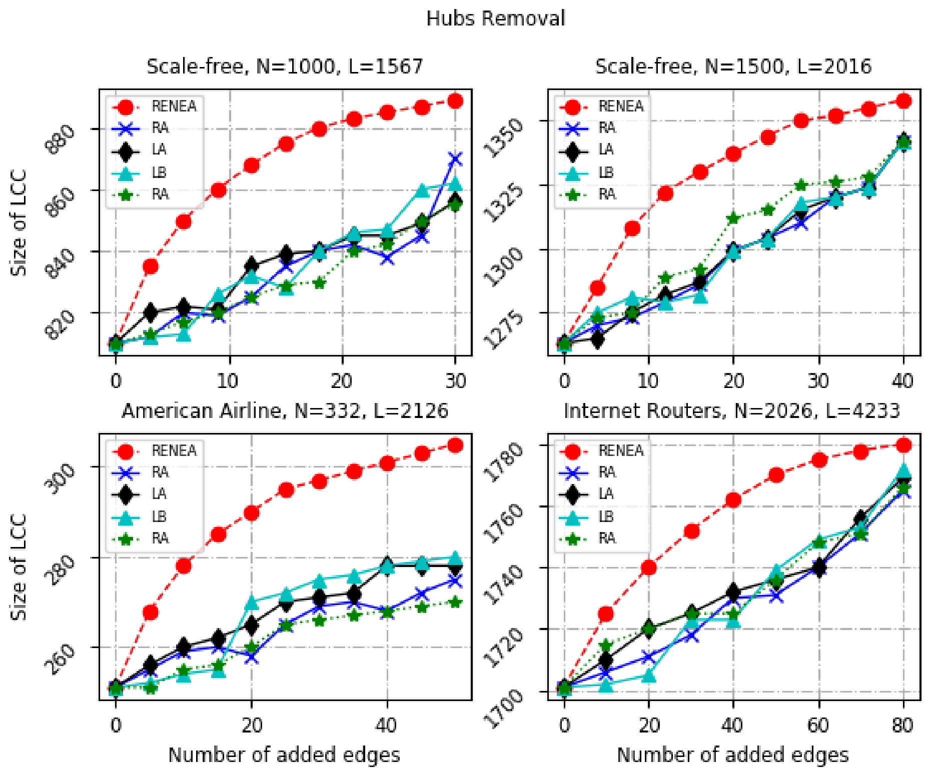


Figure 3: In this figure, x-axis represents the number of edges that are added to a network and y-axis represents the size of Largest Connected Component (LCC) after removal of hubs. The performance of RENEA against Random Addition algorithm (RA), Low-degree Addition algorithm (LA), Low Betweenness algorithm (LB), Algebraic Connectivity Based algorithm (ACB) is presented in this plot. In all networks, RENEA outperforms the other four algorithms, i.e. networks' elements stay more connected in case of attacks on hubs.

networks. Best recorded results are presented in [*Table 1*], where it can be seen that the robustness of networks improves by more than 60%, while the overall efficiency decreases by less than 3%.

6. CONCLUSION

Real-life scale-free networks are known to be resilient in terms of random failures yet fragile in the terms of targeted attacks. The consequence of hub-removal in

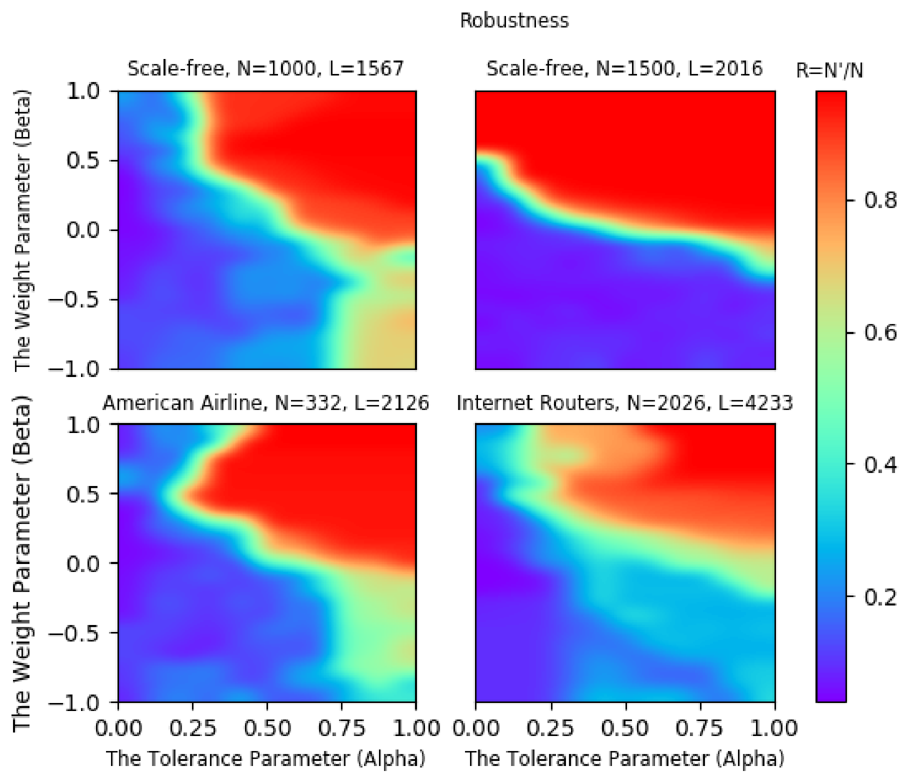


Figure 4: The network robustness as a function of Alpha and Beta. In this figure, x-axis represents the tolerance parameter (Alpha), y-axis represents the weight parameter (Beta), and the red colors show the area of high robustness.

a scale-free network is twofold. First, it can lead to existence of invalid connections in the network; and secondly, it can create traffic overload. A vertex can be deleted from a network in two ways: 1) if the vertex is attacked and thus disabled along with all its connections, and 2) if the vertex is present, but not connected to any vertices in the largest connected component (i.e., having invalid connections). The second case happens when the removal of hubs leaves many present vertices with invalid connections. On the other hand, when a hub is attacked, nearby vertices must compensate for the failed component. This, in turn, can overload these vertices, causing their failure and propagation of cascading failures throughout the network. The global cascades occur if 1) the network exhibits a highly heterogeneous degree and 2) the removed vertex is

Table 1: The performance of RENEA on different networks with *N* vertices and *L* edges. $< k >$ represents the average degree in a network. *R* and *Eff* represent the obtained robustness and efficiency after applying RENEA and edge-weight assignment. *m* indicates the maximum percentage of added edges. We added less than 2 percent of the number of edges in the original graphs to each network.

Network	N	L	$<k>$	α	β	m	R	Eff
Scale-free	1000	1567	3.2	0.2	0.3	1.5%	53%	-3.1%
Scale-free	1500	2016	3.4	0.2	0.3	1.5%	56%	-2.3%
American Airline	332	2126	9.15	0.4	0.4	1.3%	59%	-1.3%
Internet Router	2026	4233	2.67	0.3	0.3	1.2%	89%	-2.5%

among those with higher degrees. If a vertex has a relatively small load, its removal will not cause major changes in the load balance, and subsequent overload failures are unlikely to occur. However, when the load at a vertex is relatively large, its removal is likely to significantly affect loads at other vertices and possibly start a sequence of overload failures. Hence, we proposed RENEA, an iterative algorithm that is designed to enhance the overall robustness of a scale-free network against malicious attacks and mitigate the cascading failures by adding a limited number of edges to it and assigning weights to the edges. We also presented a user interface for RENEA that allows the user to be involved with making cost-efficient decisions and actually see the resulting graph. During each iteration, RENEA suggests the best edges such that adding them can increase the robustness of a graph the most, and the user can accept or reject them. Finally, the algorithm assigns a weight to each edge in order to decrease the loads on hubs in such a way that the network still can retain its efficiency. Further studies to better understand the complicated intrinsic properties of cascading failures and reliability assessment of scale-free networks are worth pursuing. Additionally, it has been shown that [32] the betweenness centrality can be approximately computed for large scale graphs in a local manner with a low computational cost. In the current version of our application we have not used this local method of computing betweenness centrality and this is something that we would like to investigate more in the future work to improve the scalability of our work.

REFERENCES

[1] Shin, Seung-Youp, and Akira Namatame. "Evolutionary optimized networks and their properties." International Journal of Computer Science and Network Security 9.2 (2009): 4-12.

[2] Cohen, Reuven, and Shlomo Havlin. "Scale-free networks are ultrasmall." Physical review letters 90.5 (2003): 058701.

[3] Barabasi, Albert-Laszlo, and Reka Albert. "Emergence of scaling in random networks." science 286.5439 (1999): 509-512.

[4] Albert, Reka, Hawoong Jeong, and Albert-Laszlo Barabasi. "Error and attack tolerance of complex networks." nature 406.6794 (2000): 378.

[5] Motter, Adilson E., and Ying-Cheng Lai. "Cascade-based attacks on complex networks." Physical Review E 66.6 (2002): 065102.

[6] Li, Shudong, et al. "Identifying vulnerable nodes of complex networks in cascading failures induced by node-based attacks." Mathematical Problems in Engineering 2013 (2013).

[7] Fan, Youjiang, et al. "Analysis of cascading failure of circuit systems based on load-capacity model of complex network." Reliability Systems Engineering (ICRSE), 2017 Second International Conference on. IEEE, 2017.

[8] Quang, Hoang Anh Tran, and Akira Namatame. "Mitigation of cascading failures with link weight control."

[9] Barabasi, Albert-Laszlo, and Zoltan N. Oltvai. "Network biology: understanding the cell's functional organization." Nature reviews genetics 5.2 (2004): 101.

[10] Motter, Adilson E. "Cascade control and defense in complex networks." Physical Review Letters 93.9 (2004): 098701.

[11] Tran, Hoang Anh Q., and Akira Namatame. "Improve Network's Robustness Against Cascade with Rewiring." Procedia Computer Science 24 (2013): 239-248.

[12] Schaffter, Thomas, Daniel Marbach, and Dario Floreano. "GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods." Bioinformatics 27.16 (2011): 2263-2270.

[13] Callaway, Duncan S., et al. "Network robustness and fragility: Percolation on random graphs." Physical review letters 85.25 (2000): 5468.

[14] Watts, Duncan J. "A simple model of global cascades on random networks." Proceedings of the National Academy of Sciences 99.9 (2002): 5766-5771.

[15] Barabasi, Albert-Laszlo, Reka Albert, and Hawoong Jeong. "Scale-free characteristics of random networks: the topology of the world-wide web." Physica A: statistical mechanics and its applications 281.1-4 (2000): 69-77.

[16] Beygelzimer, Alina, et al. "Improving network robustness by edge modification." Physica A: Statistical Mechanics and its Applications 357.3-4 (2005): 593-612.

[17] Xiao, S., et al. "Robustness of scale-free networks under rewiring operations." EPL (Europhysics Letters) 89.3 (2010): 38002.

[18] Herrmann, Hans J., et al. "Onion-like network topology enhances robustness against malicious attacks." Journal of Statistical Mechanics: Theory and Experiment 2011.01 (2011): P01027.

[19] Schneider, Christian M., et al. "Mitigation of malicious attacks on networks." Proceedings of the National Academy of Sciences 108.10 (2011): 3838-3841.

[20] Ze-Hui, Qu, et al. "Enhancement of scale-free network attack tolerance." Chinese Physics B 19.11 (2010): 110504.

[21] Yang, Yang, et al. "Improving the robustness of complex networks with preserving community structure." PloS one 10.2 (2015): e0116551.

[22] Abedijaberi, Armita, et al. "Interactive Visualization of Robustness Enhancement in Scale-free Networks with Limited Edge Addition (RENEA)."

[23] Zhao, Jichang, and Ke Xu. "Enhancing the robustness of scale-free networks." Journal of Physics A: Mathematical and Theoretical 42.19 (2009): 195003.

[24] Wu, Zhi-Xi, and Petter Holme. "Onion structure and network robustness." Physical Review E 84.2 (2011): 026106.

[25] Li, Li, et al. "Enhancing the Robustness and Efficiency of Scale-free Network with Limited Link Addition." KSII Transactions on Internet & Information Systems 6.5 (2012).

[26] Sydney, Ali, Caterina Scoglio, and Don Gruenbacher. "Optimizing algebraic connectivity by edge rewiring." Applied Mathematics and computation 219.10 (2013): 5465-5479.

[27] Liu, Jing, et al. "A comparative study of network robustness measures." Frontiers of Computer Science 11.4 (2017): 568-584.

[28] Li, Cong, et al. "Correlation between centrality metrics and their application to the opinion model." The European Physical Journal B 88.3 (2015): 65.

[29] Moussawi, Alaa, et al. "Limits of predictability of cascading overload failures in spatially-embedded networks with distributed flows." Scientific reports 7.1 (2017): 11729.

[30] Kinney, Ryan, et al. "Modeling cascading failures in the North American power grid." The European Physical Journal B-Condensed Matter and Complex Systems 46.1 (2005): 101-107.

[31] Zhao, Liang, et al. "Onset of traffic congestion in complex networks." Physical Review E 71.2 (2005): 026125.

[32] Ercsey-Ravasz, Maria, and Zoltan Toroczkai. "Centrality scaling in large networks." Physical review letters 105.3 (2010): 038701.

V. MOTIF-LEVEL ROBUSTNESS ANALYSIS OF POWER GRIDS

Armita Abedijaberi and Jennifer Leopold

Department of Computer Science

Missouri University of Science and Technology, Rolla, Missouri 65401

Motifs are often called the building blocks of networks and can be used for better understanding of the local structure of networks. Certain motifs may occur more frequently in one network than in another network with the same global structural properties such as the same nodal degree distribution. On that note, analysis of motifs is found to be an invaluable tool to show that networks with similar global topological properties may be completely different in terms of local topological properties. Most studies on robustness of power system networks against targeted attacks tend to only focus on global topological measures of power grids. However, the impact of local structures can also unveil hidden mechanisms behind vulnerability of power grids and their dynamic response to failures. In this paper, we present a preliminary study to investigate the local structure of European power grids and find the correlation between their robustness against targeted attacks and the presence or absence of certain motifs.

1. INTRODUCTION

Critical infrastructures play a pivotal role to ensure the smooth functioning of modern day living. Power grid systems are among the most critical infrastructures that are subject to numerous perturbations ranging from the common failures (e.g., breakdown of the aging generators, transformers, etc.) to the security risks and exposure of the networks to natural or man-made disasters (e.g., earthquakes, flooding, vandalism, etc.). Hence, the resistance and vulnerability have become important considerations during the design and operation of such systems. The fragility is a feature of significant interest in most complex networks, from the Internet to the genome [33]. Specifically, complex networks are often characterized by a considerable resilience against random removal or failure of individual components. However, they experience severe damages when the important components are attacked (disabled). Such targeted attacks have dramatic structural effects, typically leading to network fragmentation [32]. Most prior work on the analysis of robustness in power grids has focused on their global topology by studying their degree distribution. For instance, we know that high-degree nodes, also known as hubs, contribute to the overall higher connectivity of the network, yet they also serve as bottlenecks whose elimination leads to the fragmentation of the network [14] [25]. While studies of global structural properties of these networks have identified important issues that affect robustness, functioning in these networks typically starts within local sub-networks [35]. Herein we present a study to understand and unveil the dependencies between the local structural properties and overall robustness of European power grids. This study can be beneficial to improve the analysis and design of power grid networks. This paper is organized as follows. In Section II some preliminary topics are presented. In Section III we introduce and discuss the global characteristics of power grid networks in our dataset. Section IV focuses on experiments and results. Finally, conclusions and future work are presented in Section V.

2. PRELIMINARIES

The topology of a power grid network can be represented as a graph $G = (V,E)$, where $V = \{1,2,...,N\}$ is the set of N nodes, and $E = \{e_{ij} | i, j \in V, i \neq j\}$ is the set of M edges. In this paper, undirected and unweighted networks are considered. The connectivity or degree of a node u is the number of edges incident to u. We assume that there are no multi-edges (i.e., two or more edges that are incident to the same two vertices) or self-loops (i.e., an edge from a vertex to itself) in the input graph, and G is connected (i.e., there exists at least one path between any pair of nodes). A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ (i.e., $G' \subseteq G$), if $V' \subseteq V$ and $E' \subseteq E$. Two graphs $G' = (V', E')$ and $G'' = (V'', E'')$ are isomorphic, denoted as $G' \cong G''$, if there exists a one-to-one structure-preserving bijection between the node sets of G' and G'', $f : V' \rightarrow V''$ such that any two nodes u and v are adjacent in G' if and only if the nodes $f(u)$ and $f(v)$ are adjacent in G''.

2.1. NETWORK MOTIFS

A motif is a subgraph (pattern) that recurs in multiple parts of a network. These recurrent patterns are considered as building blocks of networks, and different combinations of a small number of motifs can generate enormously diverse forms. Studies show that the evolving biological robustness of gene regulatory networks is attributed to the occurrence of statistically significant subgraphs known as motifs [8] [7] [9]. However, despite active investigations and rich literature about motifs in systems biology and biological networks, motifs are less explored in other type of networks. Existence of network motifs implies that certain structures reflect the intrinsic properties of relations or confer functional importance of the underlying systems, thus over-presented in the networks [10]. Typically, motifs are small connected graphs with a specific structure. In many cases, the graphs on three and four nodes are selected as motifs, where two selected motifs should not be isomorphic. [Figure 1] shows all connected 3-node and 4-node motifs.

2.2. ASSESSING STATISTICAL SIGNIFICANCE OF MOTIFS

The motif signature is the number of occurrences of a motif in the network. More specifically, an occurrence of a motif is a set of nodes in the network such that the subgraph induced by this node set is isomorphic to the motif [11]. To detect recurrent motifs in a graph, it is necessary to find the signature or the number of occurrences of the motifs in the graph. However, the motif counts by themselves do not provide enough information

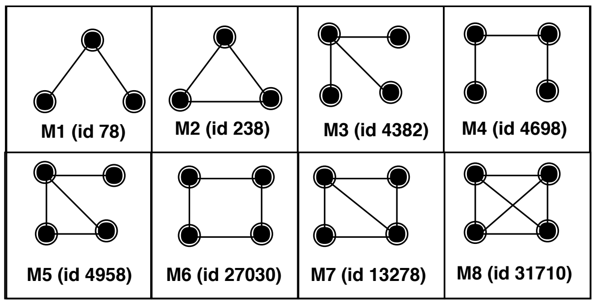


Figure 1: Eight motifs on 3 and 4 nodes, respectively. Numbers in parentheses are the Motif Identification numbers [29].

regarding the significance of motifs [36]. Hence, statistical hypothesis tests are used to determine whether a motif is significant. In fact, a null hypothesis is defined and tested to check whether there is enough evidence to reject the null hypothesis. In motif discovery, the null hypothesis claims that the given pattern is insignificant (i.e., the actual motif count is similar to the expected one). Network motifs are small connected sub-graphs occurring at significantly higher frequencies in a given input network compared with a null model [2]. Two common null models have been employed in a wide range of publications. The first null model does not change the size and average degree of nodes in the graph and generates a random Erdos-Renyi graph accordingly [3]. This null model retains the global characteristics of the network (same number of nodes and edges) but distorts the properties of individual nodes (e.g., the connectivity of each node).

The second null model randomly swaps edges from the input network [4]. A swap on an edge pair with distinct nodes (e.g., converting *AB* and *CD* into *AD* and *CB* (or *AC* and *DB*)) does not alter the connectivity of any nodes. Hence, the perturbed (null) network retains the global properties (network size, average density, and degree distribution) as well as nodal properties (connectivity of individual nodes) of the input network. Despite the advantages of the second null model, edge swaps alone may not preserve the statistics of lower-order motifs in the given network. That means in order to prove the significance of a *m*-node motif, we have to exclude the contribution from motifs with less than *m* nodes.

For instance, in a network where triangles (3-node motifs or M2) are over presented, the probability of observing any 4-node motif containing one or more triangles (e.g., M5, M7, and M8) is also elevated compared to edge-swapped null models.

To incorporate the effects of lower-order motif statistics, only the edge swaps that preserve the number of lower order motifs are accepted [5]. In simple words, the number of lower order motifs should remain the same before and after the edge swaps. An approximate counting of motifs is used [5] when the exact counting of motifs in large and dense networks is intractable and time-consuming. Let a k-node motif g_k occur f_{input} times in the input network. Let $\bar{f}_{null}$ and σ^2_{null} be the mean and variance of frequencies of g_k in a sufficiently large set of null networks, respectively. The z-score is calculated as follows:

$$z(g_k) = \frac{f_{input} - \bar{f}_{null}}{\sqrt{\sigma^2_{null}}}. \tag{1}$$

The p-value represents the probability that a motif will appear an equal or greater number of times in a null network than in the given input network. A motif is usually regarded as ‘statistically significant’ if the associated p-value < 0.01 or z-score > 2.0. The concentration of a candidate motif denotes how frequent the motif is compared to other subgraphs of the same size in the network [28]. Specifically, if there are n subgraphs of size k in the network, the concentration of the i-th size-k subgraph $g_{k,i}$ is defined as:

$$C(g_{k,i}) = \frac{f_{k,i}}{\Sigma_{j=1}^{n} f_{k,j}}, \tag{2}$$

where $f_{k,i}$ denotes the frequency of $g_{k,i}$ in the network.

2.3. CENTRALITY METRICS

The attack on nodes of a network can be random or malicious. In random attacks, nodes are removed with the same probability, whereas in malicious attacks the most important nodes are removed first.

In most existing studies on malicious attacks, a High Importance Attack (HIA) is used. In a HIA, the importance of nodes is calculated and the most important node is removed. The node degree centrality can be used to measure the importance (centrality) of nodes. The degree centrality of a node (d_i) is equal to the number of connections node i has to other nodes. The higher the degree centrality of a node, the more important that node is. In this study, we also introduce a new node centrality metric; the motif centrality of a node (m_i) is defined as the number of motifs in which node i participates. Likewise, we rank the importance of nodes based on the value of their motif centrality. The higher the motif centrality of a node, the more important that node is.

2.4. MALICIOUS ATTACKS

There are two approaches that are typically practiced in regard to network attacks: non-adaptive (or simultaneous) attacks and adaptive (or sequential) attacks. In the first approach, the list of attacked nodes is generated only once, before the node removal procedure starts [12]. In the second approach, the list of target nodes is updated after each deletion by re-calculation of the centrality index of nodes [13].

Consequently, adaptive attacks demand more processing time, but on the other hand they usually cause more damage per node removal when compared to the non-adaptive approach. The reason is, if the list of attacked nodes is measured only once, the method cannot account for the changes in the centrality order due to the removal of elements.

Thus, in the worst case the adaptive version of a procedure is as good as the non-adaptive approach, however it is generally better. In this paper we study the robustness of power grids against two types of adaptive malicious attacks as follows:

- Node Attack based on Degree Centrality.
- Node Attack based on Motif Centrality.

where in each of these adaptive attacks, first the importance of nodes is calculated and the current most important node is removed. Then, the importance of remaining nodes is recalculated, and the current most important node is removed. This process is repeated until only isolated nodes are left in the network.

2.5. TOPOLOGICAL ROBUSTNESS METRICS

The function and performance of power grids rely on their structural robustness. Robustness is the ability of a network to continue performing well when it is subject to failures or attacks. In order to decide whether a given network is robust, a metric to quantitatively measure network robustness is required. Once such a measure has been established, it is feasible to compare networks in terms of their robustness. Intuitively, robustness is focused on the alternative paths amongst components of a system, but it is a challenge to capture this concept in a mathematical formula. Therefore, a broad range of robustness metrics have been proposed. One group of robustness metrics is designed based on the concept of graph connectivity, and the ability of the graph to retain a certain degree of connectedness under node removal; these metrics include: efficiency [16], average shortest-path length [17], diameter [18], clustering coefficient [19], node and edge connectivity [20], average node betweenness [21], relative size of largest connected component [27], and percolation threshhold [4]. The other group of robustness metrics is designed based on the eigenvalue of a Laplacian matrix of a network; these methods include: algebraic connectivity [22] [1], number of spanning trees [23], and effective resistance [24]. These measures, however, are topological measures that tend to ignore any processes running on top of the network. That is, topological robustness metrics fail to take into account the operational specifications of a system and the level and quality of the service provided to the user of the system. Hence, a thorough reliability analysis of a network may not be viable using only topological robustness metrics. On that account, to evaluate the robustness of power grid networks, a topological robustness metric along with non-topological reliability measures have been used. These metrics are explained in detail in the following section.

3. EUROPEAN POWER GRIDS

In this paper we study the electricity transmission networks of four European countries (Germany, Italy, France, and Spain) where nodes represent the power station/substations or generators and edges represent the transmission lines. This dataset was obtained from the Union for the Coordination of Transmission of Electricity (UCTE) [15].

3.1. EUROPEAN POWER GRIDS ROBUSTNESS

The global characteristics of the European Power grid networks in the dataset have been previously analyzed [14], revealing a very interesting set of common regularities: (a) most of these networks are small worlds (i.e., there tends to be a path between any pair of nodes that involves only a few edges) and the largest networks display a clustering coefficient much larger than expected from the random version of the networks analyzed; (b) these networks are very sparse, meaning that the average number of edges $\langle k \rangle$ in a network with N nodes is such that $\langle k \rangle \ll N$; (c) all European power grids have exponential cumulative degree distributions. That is, the probability $p(k)$ of having a node linked to k or more other nodes follows

$$p(k) = C\,exp(\frac{-k}{\gamma}), \tag{3}$$

where C is a normalization constant and γ is a characteristic parameter. Table 1 provides a summary of the basic topological features exhibited by these four power grids.

To analytically evaluate the robustness of these networks, the percolation threshold metric [4] has been used in order to find the critical fraction of each network against selective removal of nodes.

This study [15] shows a significant deviation from predicted critical fraction values for power grids with an exponent $\gamma < 1.5$. For these networks, the real critical fraction is

Table 1: The global characteristics of two robust and two fragile European power grids are presented. These power grids are ordered by increasing γ (the exponential degree distribution characteristic parameter). N, M, and $\langle k \rangle$ represent the number of nodes, the number of edges, and the mean nodal degree, respectively.

Group	**Country**	N	M	$\langle k \rangle$	γ
Robust	Germany	445	560	2.51	1.237
	Italy	272	368	2.70	1.238
Fragile	France	667	899	2.69	1.895
	Spain	474	669	2.82	2.008

higher than the theoretical one for the same γ. This suggests an increased robustness for these networks compared to those with $\gamma > 1.5$. This observation divides these power grids into two classes of *robust* $(\gamma < 1.5)$ and *fragile* $(\gamma > 1.5)$ networks.

In order to evaluate the real existence of these two classes of networks, real non-topological reliability measures have also been considered: (1) energy not supplied; (2) total loss of power; (3) average interruption time, which is the ratio between total energy not supplied and the average power demand per year.

Results suggest a positive correlation between static topological robustness and non-topological reliability measures and, as a consequence, a clear differentiation between two classes of networks in terms of their level of robustness [15].

Once the Germany and Italy power grids were experimentally classified as robust and the France and Spain power grids were classified as fragile, we wanted to investigate the impact of local structures or motifs on fragility or robustness of such networks. Our goal was to study the concentration of motifs in the power grids and analyze the decaying rate of motif concentration under different types of attacks. This would enhance our understanding of the role of local structures in global robustness properties of the corresponding network.

4. EXPERIMENTS

We implemented the motif finder algorithm that was explained in part C of section II and calculated the z-score (Eq. 1) of all the eight motifs [Figure 1] in these four power grid networks to determine the statistically significant motifs. Table 2 shows the z-score of these motifs in each corresponding network. Motifs with z-score of 2.0 or higher are considered significant and those with z-score value of less than 2.0 are denoted as NS or not significant. Despite the overwhelming presence of motif M1, this motif is not statistically significant in any of four networks. As shown in Table 2, M8 is not considered significant in any of these networks either. The rest of the motifs are significant for all networks except that M3 is considered NS in the Spain power grid. Despite the robust/fragile category, all of these networks have almost the same significant motifs.

Table 2: The z-score as a qualitative measure of statistical significance of each motif for each network is presented in this table. NS denotes that the motif is not significant.

Country	**Z-score**							
	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**
Germany	NS	16	96	67	75	2.5	2.1	NS
Italy	NS	34	134	54	34	2.7	2.3	NS
France	NS	24	87	36	14	7.9	2.1	NS
Spain	NS	57	NS	56	47	2.8	2.2	NS

To study further, we examined the concentration of the significant motifs (Eq. 2) in the networks. The concentration of M6 and M7 are very low and similar for all networks. But, there is a notable difference between level of concentration of M2, M3, M4, and M5 in the networks as shown in [Figure 2]. In [Figure 2], the x-axis represents the value of γ, which is increasing from left to right, and the y-axis represents the percentage of

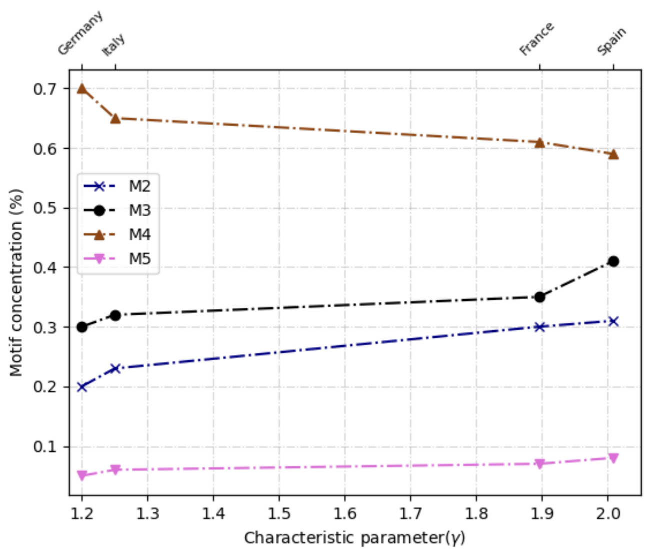


Figure 2: Motif concentration for different motifs in different networks is presented in this plot. The x-axis represents the value of γ which increases as the fragility of networks increases.

motif concentration, which is increasing from bottom to top. As shown in this figure, the concentration of M4 is decreasing as the value of γ is increasing. On the other hand, the concentration of M2, M3, and M5 are increasing as the fragility of the networks increases with γ.

In spite of having similar significant motifs, networks in the robust group (Germany and Italy) demonstrate a different level of concentration of certain motifs compared to networks in the fragile group (France and Spain). As seen in [Figure 2], fragility seems to increase as the elements of power grids become more interconnected and motifs such as stars (M3) and triangles (M2 and M5) began to appear.

We then computed the *z*-score for the larger significant motifs, with more than 4 nodes, in the networks. [Figure 4] shows two significant 5-node motifs (M9 and M10) that were found in the Germany and Italy power grids. However, these two motifs are not statistically significant in the Spain and France power grids. No significant motifs with more than 5 nodes were found in any of these networks. These results suggests that

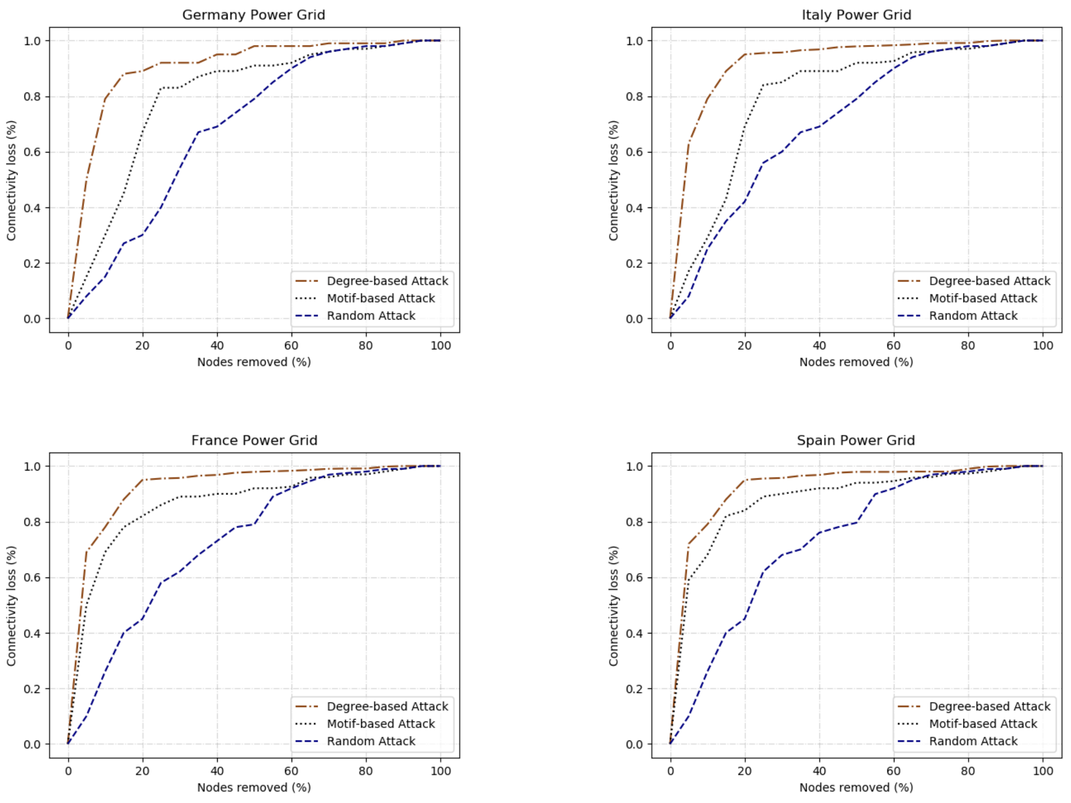


Figure 3: The percentage of connectivity loss vs. the percentage of node removal in different types of node attacks.

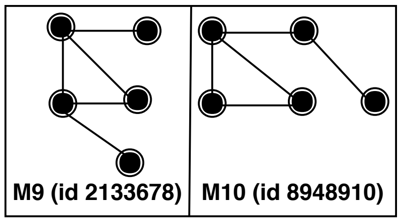


Figure 4: The 5-node motifs that are only significant in the Germany and Italy power grids. Numbers in parentheses are the Motif Identification numbers [29].

even in networks that share similar small motifs, as the motif size increases, differences arise. We next compared the behavior of networks in the robust group versus the networks in the fragile group against different types of adaptive malicious attacks. We simulated three types of attacks against all four networks, namely Degree-based attack, Motif-based attack, and Random attack. For each attack we measured the percentage of the connectivity loss for a different percentage of node removal. Connectivity Loss is a purely topological measure of impact that a power grid encounters when some nodes are removed from the system [34]. Using this measure we can calculate, for example, how much connectivity is lost in terms of how many generators a substation can access after removing a node from the system. In essence, the connectivity loss measures the decrease of the ability of distribution substations to receive power from the generators.

In the unperturbed state each distribution substation can receive power from any of the N_g generators. As nodes are removed, the number of generators connected to a certain distribution substation i, N_g^i , decreases. The connectivity loss C_L is used to quantify the average decrease in the number of generators connected to a distributing substation and is defined as,

$$C_L = 1 - \left\langle \frac{N_g^i}{N_g} \right\rangle_i , \tag{4}$$

where the averaging is done over every distributing substation. [Figure 3] shows the increase in the connectivity loss in different networks against various types of attacks. As shown, for all the countries, the degree-based attack causes the steepest curve for connectivity loss, whereas the random attack has the slightest incline in curve for connectivity loss. This means the degree-based attack causes the connectivity loss to increase more rapidly than the other types of attacks. For all the networks the motif-based attack curve falls between the degree-based and random attack curve. It can noticeably be seen that the gaps between the connectivity loss curves of degree-based and motif-based attacks tend to be narrower in the France and Spain power grid networks. This observation suggests that there is a correlation between the degree centrality and motif centrality of nodes in the fragile network.

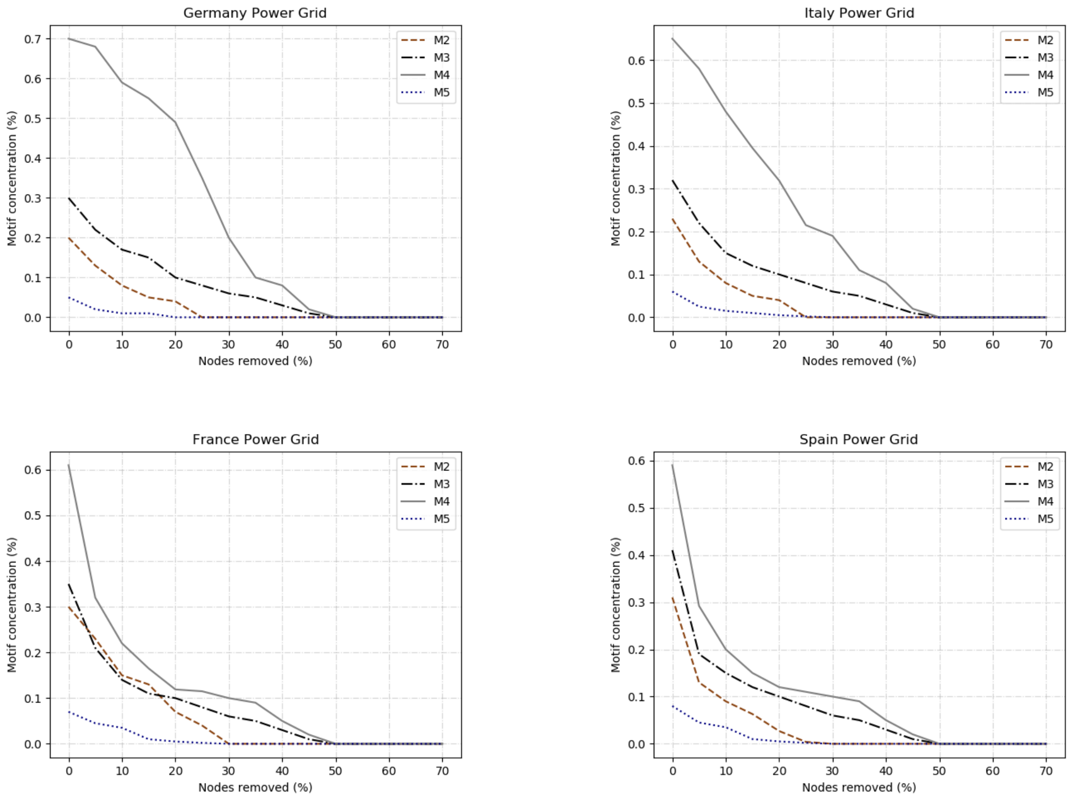


Figure 5: Motif concentration decay for significant motifs vs. the node removal percentage in degree-based node attacks.

Furthermore, [Figure 5] shows the decay in concentration of different significant motifs in different networks against the degree-based attack. In this figure, the x-axis represents the percentage of node removal and the y-axis represents the motif concentration for various motifs. We simulated a degree-based attack as this attack has been shown to be the most damaging attack. At the beginning M4 has the highest concentration in all of the networks. However, its concentration decreases with different slope in the different networks as more nodes are removed. The slope of M4 concentration decay in the France and Spain power grid networks is steeper than the Germany and Italy networks. The wide gap between the M4 and other motifs in the Germany and Italy power grids suggests that the degree centrality and M4-centrality values are highly correlated in these two networks. As far as the concentration decays of other motifs are concerned, no interesting patterns are discernible. Lastly, we tested the evolution of z-score value of different significant motifs when the edges of the network are removed. One of the most common measures of the importance of an edge is its betweenness, i.e. the number of shortest paths passing through that edge (normalized in the case where multiple shortest paths between some vertices occur). More precisely, the betweenness centrality of an edge e, (b_e), is defined as:

$$b_e = \sum_{i \neq j \in V} \frac{\sigma_{ij}(e)}{\sigma_{ij}}, \tag{5}$$

where σ_{is} denotes the total number of shortest paths between nodes i and j, and $\sigma_{ij}(e)$ denotes the total number of shortest paths between nodes i and j that pass through edge e. The higher the betweenness centrality of an edge, the more important that edge is.

[Figure 6] shows the z-score value of significant motifs as a function of the percentage of removed edges in a random-based attack and a betweenness-based attack in all four networks. In the betweenness-based edge attack, we used an adaptive approach; starting from each network we found the candidate edge (i.e., the edge with highest betweenness centrality value) to remove, then we updated the betweenness centrality values for all the remaining edges and found the next candidate to remove. In the random-based attack, one edge randomly was removed during each iteration of the attack. The z-score of different motifs were calculated with respect to corresponding edge removal percentage and results were plotted in [Figure 6]. As shown, for motifs M3, M4, and M5 the random-based edge attack and betweenness-based edge attack behave similarly. Both attacks cause an increase in the value of z-score of M3 and M4. However, they cause a decrease in the z-score value of M5. One interesting result is related to the z-score value of M2, where these two attacks behave differently. In the case of the betweenness-based edge attack, the z-score value of M2 in the Spain power grid increases and in the France power grid it remains almost the same. Whereas, when a random-based edge attack happens for both of these networks, the

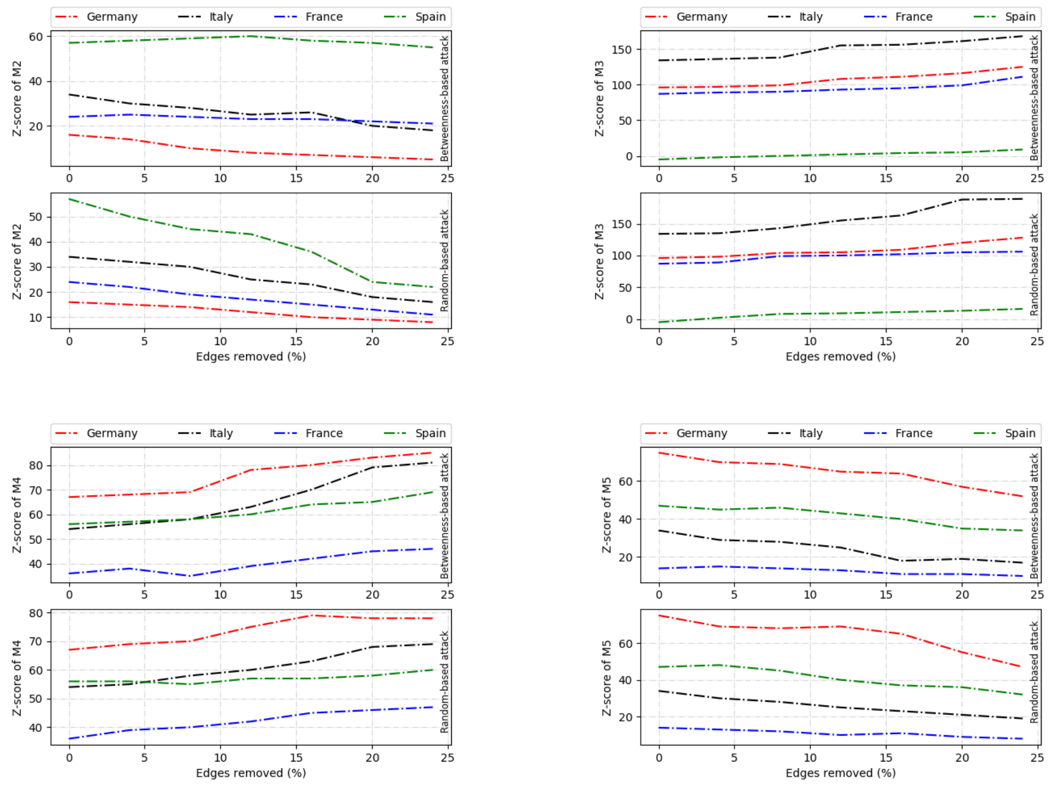


Figure 6: Z-score of significant motifs as a function of the percentage of removed edges in random-based vs. betweenness-based edge attack.

z-score value of M2 decreases. On the other hand, for the Germany and the Italy power grids the *z*-score value of M2 decreases similarly in both attacks. This behavior can be due to the fact that edges with high-betweenness centrality in the France and the Spain power grids are probably bridges that are connecting different parts of these networks.

5. CONCLUSION

In this paper we presented a preliminary study on the robustness of power grids and their sub-structures, namely motifs. Network motifs are statistically overrepresented sub-structures (subgraphs) in a network, and their detection has recently become an important part of network analysis across a variety of disciplines. Network motifs are also referred to as the building blocks of complex networks. This is because these small building blocks fit together in a specific way to give a network a variety of properties that result in stability or resiliency under certain conditions.

In this study we used four European power grids as our dataset. Using the topological and non-topological robustness metrics, these networks have been categorized as robust and fragile. By calculating the z-score and concentration values of different motifs with three and four nodes, we identified the significant motifs in these networks. According to the results, all of these four networks have similar significant motifs with different levels of concentration. Additionally, the location of motifs in a network can contribute to the overall robustness of the network. Our results revealed that networks with similar small motifs can have different larger motifs. This suggests that as motif size increases, the differences between networks arise whereby larger motifs can define unique structures. One of the main goals of researching network motifs is to gain insight into how the aggregate of small group interactions forms the macroscopic behavior we see in complex networks. Attacks on power grids are no longer just a theoretical concern and we believe that demonstrating such insights for power grid networks may be beneficial to the design and analysis of more robust networks.

In the future, we want to further investigate on directed motifs and see how aggregation of different directed motifs can create unique structural and topological properties that can directly contribute to the overall robustness of a directed network such as the Internet. We are optimistic that these studies will shed light on the optimal design of more robust networks.

REFERENCES

[1] Shatto, Tristan A., and Egemen K. Cetinkaya. "Spectral Analysis of Backbone Networks Against Targeted Attacks." DRCN 2017-Design of Reliable Communication Networks; 13th International Conference; Proceedings of. VDE, 2017.

[2] Aittokallio, Tero, and Benno Schwikowski. "Graph-based methods for analysing networks in cell biology." Briefings in bioinformatics 7.3 (2006): 243-255.

[3] Britton, Tom, Maria Deijfen, and Anders Martin-Lof. "Generating simple random graphs with prescribed degree distribution." Journal of Statistical Physics 124.6 (2006): 1377-1397.

[4] Silva, Miguel EP, Pedro Paredes, and Pedro Ribeiro. "Network motifs detection using random networks with prescribed subgraph frequencies." Workshop on Complex Networks CompleNet. Springer, Cham, 2017.

[5] Yeang, Chen-Hsiang, Liang-Cheng Huang, and Wei-Chung Liu. "Recurrent structural motifs reflect characteristics of distinct networks." Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012). IEEE Computer Society, 2012.

[6] Dahiru, Tukur. "P-value, a true test of statistical significance? A cautionary note." Annals of Ibadan postgraduate medicine 6.1 (2008): 21-26.

[7] MacNeil, Lesley T., and Albertha JM Walhout. "Gene regulatory networks and the role of robustness and stochasticity in the control of gene expression." Genome research 21.5 (2011): 645-657.

[8] Noman, Nasimul, et al. "Evolving robust gene regulatory networks." PloS one 10.1 (2015): e0116258.

[9] Ebert, Margaret S., and Phillip A. Sharp. "Roles for microRNAs in conferring robustness to biological processes." Cell 149.3 (2012): 515-524.

[10] Gorochowski, Thomas E., Claire S. Grierson, and Mario di Bernardo. "Organization of feed-forward loop motifs reveals architectural principles in natural and engineered networks." Science advances 4.3 (2018): eaap9751.

[11] Weihe, Karsten. "Motifs in networks." Gems of Combinatorial Optimization and Graph Algorithms. Springer, Cham, 2015. 59-68.

[12] Iyer, Swami, et al. "Attack robustness and centrality of complex networks." PloS one 8.4 (2013): e59613.

[13] Holme, Petter, et al. "Attack vulnerability of complex networks." Physical review E 65.5 (2002): 056109.

[14] Rosas-Casals, Marti, Sergi Valverde, and Ricard V. Sole. "Topological vulnerability of the European power grid under errors and attacks." International Journal of Bifurcation and Chaos 17.07 (2007): 2465-2475.

[15] Sole, Ricard V., et al. "Robustness of the European power grids under intentional attack." Physical Review E 77.2 (2008): 026102.

[16] Latora, Vito, and Massimo Marchiori. "Efficient behavior of small-world networks." Physical review letters 87.19 (2001): 198701.

[17] Shannon, Colleen, and David Moore. "The spread of the witty worm." IEEE Security & Privacy 2.4 (2004): 46-50.

[18] Albert, Reka, Hawoong Jeong, and Albert-Laszlo Barabasi. "Internet: Diameter of the world-wide web." nature 401.6749 (1999): 130.

[19] Bollobas, Bela, and Oliver M. Riordan. "Mathematical results on scale-free random graphs." Handbook of graphs and networks: from the genome to the internet (2003): 1-34.

[20] Dekker, Anthony H., and Bernard D. Colbert. "Network robustness and graph topology." Proceedings of the 27th Australasian conference on Computer science-Volume 26. Australian Computer Society, Inc., 2004.

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

[22] Jamakovic, A., and S. Uhlig. "On the relationship between the algebraic connectivity and graph's robustness to node and link failures." Next Generation Internet Networks, 3rd EuroNGI Conference on. IEEE, 2007.

[23] Baras, John S., and Pedram Hovareshti. "Efficient and robust communication topologies for distributed decision making in networked systems." Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on. IEEE, 2009.

[24] Ellens, Wendy, et al. "Effective graph resistance." Linear algebra and its applications 435.10 (2011): 2491-2506.

[25] Abedijaberi, A., Eloe, N., Leopold, J., & Maryville, M. O. Interactive Visualization of Robustness Enhancement in Scale-free Networks with Limited Edge Addition (RENEA). Proceedings of the 23rd International Conference on Distributed Multimedia Systems, 2017.

[26] Callaway, Duncan S., et al. "Network robustness and fragility: Percolation on random graphs." Physical review letters 85.25 (2000): 5468.

[27] Schneider, Christian M., et al. "Mitigation of malicious attacks on networks." Proceedings of the National Academy of Sciences 108.10 (2011): 3838-3841.

[28] Wong, Elisabeth, et al. "Biological network motif detection: principles and practice." Briefings in bioinformatics 13.2 (2011): 202-215.

[29] Kashtan, Nadav and Itzkovitz, Shalev and Milo, Ron and Alon, Uri. "mfinder Tool Guide." (2002)

[30] Lee, Chang-Yong. "Correlations among centrality measures in complex networks." arXiv preprint physics/0605220 (2006).

[31] Albert, Reka, Istvan Albert, and Gary L. Nakarado. "Structural vulnerability of the North American power grid." Physical review E 69.2 (2004): 025103.

[32] Rosas-Casals, Marti. "Power grids as complex networks: topology and fragility." Complexity in Engineering, 2010. COMPENG'10.. IEEE, 2010.

[33] Dorogovtsev, Sergei N., and Jose FF Mendes. Evolution of networks: From biological nets to the Internet and WWW. OUP Oxford, 2013.

[34] Freeman, Linton C. "The gatekeeper, pair-dependency and structural centrality." Quality and Quantity 14.4 (1980): 585-592.

[35] Dey, Asim Kumer, Yulia R. Gel, and H. Vincent Poor. "Motif-based analysis of power grid robustness under attacks." Signal and Information Processing (GlobalSIP), 2017 IEEE Global Conference on. IEEE, 2017.

[36] Abedijaberi, Armita, and Jennifer Leopold. "FSMS: A Frequent Subgraph Mining Algorithm Using Mapping Sets." Machine Learning and Data Mining in Pattern Recognition. Springer, Cham, 2016. 761-773.

SECTION

2. FUTURE WORK

As a part of the future research, we intend to extend our work to mine real-life large-scale graphs. We want to scale and implement our sub-pattern mining algorithms in a distributed data processing platform to be able to handle a single large graph.

We also intend to extend our work on robustness analysis of real-life large-scale networks. In this context, our work could be carried out in the following research directions.

2.1. GRAPH MINING USING DISTRIBUTED PLATFORMS

Most of the frequent subgraph mining algorithms focus only on the transactional graphs setting, where the input graph consists of many small graphs. However, modern applications (e.g., social networks, the Internet, protein-protein interactions, etc.) typically involve a single large graph with millions of vertices.

Additionally, the set of frequent patterns and their subgraphs in a graph can be exponential in the size of the original graph, resulting in an explosion of the computation of intermediate states.

There are different data processing platforms such as Pregel [1] and Arabesque [2] that are designed for deployment of distributed graph analytic algorithms.

In particular, Pregel offers a "Think Like a Vertex" (TVL) programming paradigm, where each vertex of the input graph is a processing element holding a local state and communicating with its neighbors in the graph.

In contrast, Arabesque offers a "Think Like an Embedding" (TLE) programming paradigm, where each embedding of the input graph is a processing element.

We intend to scale our frequent subpattern mining algorithms using these platforms in order to mine frequent subpatterns in a single large graph.

2.2. MOTIF-BASED ANALYSIS OF GRAPH ROBUSTNESS

Network motifs are often called the building blocks of graphs. They are patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks. Analysis of motifs is found to be an indispensable tool for understanding local network structures and their effect on a global network topology. As a result, networks that are similar in terms of global topological properties may differ noticeably at a local level.

We intend to extend our preliminary study and investigate further on directed motifs and how they contribute to the overall robustness of a directed network such as the Internet. We also we want to know how certain significant motifs contribute to the overall robustness of networks that are more likely to experience edge failures rather than node failures. We believe that these studies can be beneficial in order to design more robust infrastructure networks.

3. CONCLUSION

This dissertation presents algorithms and methods to mine and analyze different types of networks. Firstly, an algorithm for frequent subgraph mining in a graph was proposed. Secondly, a more efficient algorithm for frequent subtree mining in a large graph along with its implementation in a distributed framework was presented. Thirdly, an algorithm to improve the robustness of graphs subject to targeted attack was proposed. Fourthly, an algorithm to mitigate the cascading failures in scale-free graphs was proposed. Lastly, the dissertation was concluded by presenting a study on the interdependencies between the global structural properties and the local topological structures of the graph.

In paper I, a frequent subgraph mining algorithm named FSMS was proposed to discover all frequent subgraphs of different sizes in an input graph, where the frequency is determined based on a predefined user input threshold. There are some bottlenecks regarding finding the frequent subgraphs in a large graph, where isomorphism checking is one of them. FSMS was designed to eliminate the process of isomorphism checking by using Mapping Sets. In paper II, a frequent subtree mining algorithm named SMAL was proposed. SMAL is an efficient extension of FSMS to significantly avoid expensive isomorphism testing and generating duplicated candidates by using automorphism sets. SMAL was also designed for distributed computing.

In Paper III an iterative algorithm named RENEA was designed to enhance the overall robustness of a scale-free network against malicious attacks by adding a limited number of edges to it. Adding new connections to the nodes arbitrarily and without any constraint can change the nodal degree of the graph and disturb other structural properties of it. RENEA is an iterative algorithm that suggests the best edges such that adding them can increase the robustness of a graph the most. In paper IV an extension to RENEA was proposed. In this paper we discussed the phenomenon of cascading failures in scale-free networks such that when a hub is attacked, nearby vertices must compensate for the failed component. This, in turn, can overload these vertices, causing their failure and propagation of cascading failures throughout the network. In order to mitigate the cascading failures, our algorithm assigns a weight to each edge of the network in order to decrease the loads on hubs in such a way that the network still can retain its efficiency.

In paper V, we presented a preliminary study on robustness of power grids and their sub-structures, namely motifs. Our goal in this study was to gain insight into how the aggregate of small group interactions forms the macroscopic behavior that we see in complex networks.

BIBLIOGRAPHY

[1] Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010.

[2] Teixeira, Carlos HC, et al. "Arabesque: a system for distributed graph mining." Proceedings of the 25th Symposium on Operating Systems Principles. ACM, 2015.

[3] Opsahl, Tore, Filip Agneessens, and John Skvoretz. "Node centrality in weighted networks: Generalizing degree and shortest paths." Social networks 32.3 (2010): 245-251.

[4] Callaway, Duncan S., et al. "Network robustness and fragility: Percolation on random graphs." Physical review letters 85.25 (2000): 5468.

[5] Jiang, Chuntao, Frans Coenen, and Michele Zito. "A survey of frequent subgraph mining algorithms." The Knowledge Engineering Review 28.1 (2013): 75-105.

[6] Barabasi, Albert-Laszlo, and Eric Bonabeau. "Scale-free networks." Scientific american 288.5 (2003): 60-69.

[7] Watts, Duncan J., and Steven H. Strogatz. "Collective dynamics of small-world networks." nature 393.6684 (1998): 440.

[8] Barabasi, Albert-Laszlo, and Reka Albert. "Emergence of scaling in random networks." science 286.5439 (1999): 509-512.

[9] Brouwer, Andries E., and Willem H. Haemers. "Distance-regular graphs." Spectra of Graphs. Springer, New York, NY, 2012. 177-185.

[10] Newman, Mark EJ. "Random graphs with clustering." Physical review letters 103.5 (2009): 058701.

[11] Barabasi, Albert-Laszlo, Reka Albert, and Hawoong Jeong. "Scale-free characteristics of random networks: the topology of the world-wide web." Physica A: statistical mechanics and its applications 281.1-4 (2000): 69-77.

[12] Newman, Mark EJ. "Modularity and community structure in networks." Proceedings of the national academy of sciences 103.23 (2006): 8577-8582.

[13] Lancichinetti, Andrea, et al. "Finding statistically significant communities in networks." PloS one 6.4 (2011): e18961.

[14] Watts, Duncan J. "A simple model of global cascades on random networks." Proceedings of the National Academy of Sciences 99.9 (2002): 5766-5771.

[15] Wang, Wen-Xu, and Guanrong Chen. "Universal robustness characteristic of weighted networks against cascading failure." Physical Review E 77.2 (2008): 026101.

VITA

Armita Abedijaberi was born in Esfahan, Iran. She received her bachelor of Computer Engineering degree in 2008 from Islamic Azad University Central Tehran Branch. She began her Ph.D. at Missouri University of Science and Technology in August 2013. In July 2018, she received her Ph.D. in Computer Science from Missouri University of Science and Technology.