4-2016

A faster version of Louvain method for community detection for efficient modeling and analytics of cyber systems

Sunanda Vivek Shanbhaq
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Sunanda Vivek Shanbhag

Entitled
A FASTER VERSION OF LOUVAIN METHOD FOR COMMUNITY DETECTION FOR EFFICIENT MODELING AND ANALYTICS OF CYBER SYSTEMS

For the degree of Master of Science

Is approved by the final examining committee:

John Springer
Chair

Eric Matson

Eric Dietz

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): John Springer

Approved by: Jeffrey Whitten 4/26/2016

Head of the Departmental Graduate Program Date

A FASTER VERSION OF LOUVAIN METHOD FOR COMMUNITY DETECTION FOR EFFICIENT MODELING AND ANALYTICS OF CYBER SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Sunanda Vivek Shanbhag

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2016

Purdue University

West Lafayette, Indiana

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. John Springer, for guiding me throughout the process of doing research. It was because of his immense faith in me that I could successfully complete my thesis.

I would also like to thank my committee members, Dr. Eric Matson and Dr. Eric Dietz for the feedback and advice they provided.

I am also thankful to Information Technology at Purdue (ITAP), West Lafayette, IN for providing the infrastructure needed to support this research.

Lastly, I would like to thank my family for supporting me throughout this journey. A special note of thanks to my friends Kiran and Harshal for being there to motivate me when I needed a push.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table Page

Appendix Table Page

LIST OF FIGURES

Figure Page

GLOSSARY

Community - A community is a set of nodes in a network that is densely connected internally.

Community detection - The ability to detect a community structure in a network is known as community detection (Girvan & Newman, 2002).

Modularity - It is a quality measure for partitions. It is a numerical value between -1 and 1, which is an index of how good the partitions are (Newman, 2002).

ABSTRACT

Shanbhag, Sunanda V. M.S., Purdue University, May 2016. A Faster Version of Louvain Method for Community Detection for Efficient Modeling and Analytics of Cyber Systems . Major Professor: John Springer.

Cyber networks are complex networks with various hosts forming the entities of the network and the communication between them forming the edges of the network. Most cyber networks exhibit a community structure. A community is a group of nodes that are densely connected with each other as compared to other nodes in the network. Representing an IP network in the form of communities helps in viewing the network from different levels of granularity and makes the visualization of the network cleaner and more pleasing to the eye. This will help significantly in cyber attack detection in large scale cyber networks. In order to serve this purpose, it is important to retrieve the community structure fast, before the damage done by the attacker spreads and compromises the system.

This research is an effort to bring about fast community detection of large cyber networks. The Louvain method, which is one of the most popular modularity optimization algorithms, is studied thoroughly and modified to make it faster, while preserving the quality of partitions at the same time.

CHAPTER 1. INTRODUCTION

This chapter gives a brief introduction to the research conducted. The introduction chapter includes a description of the problem statement, the research question, the scope of the research and its significance. It also specifies the assumptions, limitations and delimitations of this research.

1.1 Problem Statement

This research was a study based on fast community detection in large scale cyber networks for cyber security applications. Detecting communities in large scale networks helps in obtaining multiple levels of granularity, which can make detection of cyber attacks in large networks easier.

1.2 Research Question

Does the modified version of Louvain method proposed in this study perform faster than the original algorithm, along with preservation of accuracy, for large scale cyber systems?

1.3 Scope

Graph analytics has become very popular for many Big Data problems. Some of these problems include social network analysis, semantic searches, biological and chemical studies, network analysis and Cybersecurity. This study focuses on graph analytics for Cybersecurity. Cybersecurity studies involve analysis of the Internet in order to find anomalies in the network. The Internet has been growing physically, functionally and geographically (Ge, 2001). This study considers only the physical aspect of the Internet. The physical structure of the Internet shows its topology and the interaction between the network hosts, routers and servers in the form of a graph. A large IP network consists of millions of nodes and numerous edges between these nodes, indicating the traffic between the nodes. This cyber traffic analysis forms the base for analysis of Cybersecurity of the system. The traffic information can be represented by graphs, where the hosts form the vertices and the traffic between hosts forms the edges. These graphs can be used for analysis. However, many algorithms for graph analytics, which are designed for small graphs, are very inefficient while working with very large graphs. “Recent years have witnessed a substantial new movement in network research, with the focus shifting away from the analysis of single small graphs and the properties of individual vertices or edges within such graphs to consideration of large-scale statistical properties of graphs” (Newman, 2003, p167). The use of multi-scale graphs makes the analysis of large-scale networks easier. Multi-scale graphs are traditional graphs paired with a

hierarchical partition of its nodes (Hogan, Hui, Choudhury, Halappanavar, Oler, & Joslyn, 2013).

This study worked on one community detection algorithm, the Louvain method, to make it work faster on large networks along with preserving the quality of communities detected.

1.4 Significance

The Internet is a global network connecting millions of computers. Individuals and organizations can connect to any place on the network at any time, irrespective of the geographic boundaries. Global organizations have large cyber networks, which are used to store and transfer data globally. Most of this data is confidential. It includes personal information of the customers and employees as well as confidential business data or intellectual property of the organization. While the Internet has made life easier and more fun for most individuals and organizations, it has also led to an increase in their vulnerability to attack or intrusion. According to the Ponemon Institute that annually provides cross-country and cross-industry information, security and data breaches resulted in an average financial impact of US$9.4 million in 2013 and the numbers will increase significantly in the future years (Biener, Eling, & Wirfs, 2015). A report prepared for the World Economic Forum estimates total economic losses from cybercrime in 2009 in the United States alone at more than US$500 million (Biener, Eling, & Wirfs, 2015). Cyber space is vulnerable to many attacks and risks. Breach of such data may lead to severe problems. Cyber attacks can range from a minor attack on

an individual to major attacks on big organizations, leading to major losses. The attacks that target government and critical infrastructure can become a national security issue. Hence, it is very important to detect attacks. Also, it is important to identify the attacks before the damage done by them goes beyond control. Fast detection of attacks helps in controlling the damage done by the attacks on the network.

Cybersecurity has become a topic of utmost importance lately. Due to the increasing popularity of the Internet, the amount of communication and transfer of data over the network has been increasing at a very high rate. Cybersecurity analysts have to work with a large number of packets in order to analyze large networks for cyber attack detection.

Graph analytic techniques are widely used on cyber traffic data. However, these techniques are computationally intensive, mainly because of the need to do the analysis on such a large number of nodes. Hence, graph partitioning techniques are used in order to divide the graphs into subgraphs that together represent the whole graph. Each subgraph is known as a supernode. A graph is formed using these supernodes and analytics is conducted on this smaller shrunken graph. Clustering of nodes of the graph to form a supernode can help solve the issue mentioned above in two ways. Firstly, it will help in detecting communities in the network. Each community will represent a group of nodes that have high interaction with each other. A graph of communities will let cyber defenders analyze the network at different levels of granularity (Hogan et al., 2013). This will make the detection of an attack and victims affected easier. Secondly,

efficient visualization of large networks makes analysis significantly easier. A clustering algorithm finds subgraphs, each of which is then replaced with a meta-node as a cluster. In this way, a coarse graph is obtained by replacing all subgraphs with their corresponding meta-nodes (Huang, X. & Huang, W., 2015). A graph that has less number of nodes and is representative of the original graph makes it visually more understandable. One of the requirements of visualizations is that they should aid in real-time analysis (Huang, et al., 2015). Hence, using a clustering algorithm that is slow won't be very useful. A fast community detection algorithm will provide communities faster, which in turn will lead to a faster analysis.

1.5 Assumptions

The assumptions of this study were:

1. The hardware used for implementing the algorithms was constant and worked with equal efficiency and reliability for each algorithm.
2. The dataset used was an anonymized dataset. The loss due to data anonymization did not affect the results.
3. According to Girvan and Newman (2004), values of modularity for networks with a robust community structure fall within the range of 0.3 and 0.7. For this study, if the modularity value is above 0.5, the quality of partitions made by the algorithms was considered good.

1.6 Limitations

The limitations of this study were:

1. As this study focused on the runtime of the algorithm, it could have led to loss of precision to some extent.
2. As the dataset is anonymized, the hardware specification of the components of the network is not known.

1.7 Delimitations

The delimitations of this study were:

1. This study did not include any work on intrusion detection techniques. It focuses only on graph partitioning techniques.
2. This study did not solve the general problem. It addresses only a part of it.
3. Even though the results of this study might be useful in other areas, this study focused only on requirements of cybersecurity applications.
4. This study considered community detection of static networks only. Community detection in dynamic networks was out of the scope of this research.
5. This study considered only modularity and the Louvain algorithm. It does not include other metrics or algorithms for comparison.
6. This study used only source IP and destination IP address for community

detection. Consideration of port numbers or any other characteristic of the network was out of scope.

1.8 Summary

This chapter introduced the research being conducted by the researcher. Along with the scope and significance, the assumptions, limitations and delimitations of the study were mentioned too.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

This chapter gives details about the relevant literature that was studied by the researcher in order to strengthen the theoretical framework for this research. Please note that the use of symbols and formulas followed the convention used by the authors of the relevant literature. This thesis attempted to cite cases where the notation was adopted from other sources.

2.1 Approach to this Review

The literature review starts with explanation of a few commonly occurring cyber attacks, which can be detected by analyzing the network traffic and looking for anomalous behavior in it. Further, it gives a description of cyber networks in the form of graphs. This is followed by explanation of community detection and the graph metrics and algorithms used to detect communities in networks.

2.2 Cyber Attacks

As the Internet grows, failures and attacks can cause damage on a significantly larger scale. Computers and devices are connected to each other in the network. An attack on a device can propagate the effect of the attack to the devices it is connected

to, hence degrading the performance of the network as a whole (Park, Khrabov, Pennock, Lawrence, Giles & Ungar, 2003). Most attacks are characterized by anomalous behavior over the network. An unusual traffic pattern observed in a network could indicate the presence of an attack in order to transmit data to unauthorized destinations (Ahmed, Naser Mahmood, & Hu, 2016).

2.3 Cyber Networks as Graphs

As mentioned in section 2.2, large scale cyber networks include hosts and the communication between them. This information can be best described in the form of a graph.

In simple mathematical terms, a graph is the representation of a set of entities (vertices) and a set of connections between these entities (edges). The graph theory has vast applications in many research areas. Graphs are extensively used in chemistry. According to Balaban (1985), graph theory, especially the concept of isomorphism, provides a strong basis for studies in chemistry. Graph coloring is extensively used in scheduling applications, like job scheduling, aircraft scheduling, etc. Graphs also represent social networks very efficiently. Network security is another important application of graph theory in the field of computer science. Other applications include efficient routing, information networks and many others. This study focuses on using graphs to make network traffic analysis for cybersecurity more efficient.

The relationships between entities, represented by edges, describe a lot about the network. Analysis of these relationships leads to detection of various activities and

events happening over the network that wouldn't have been detected by analyzing entities alone (Bliss & Schmidt, 2013). Anomalous behavior of nodes with other nodes of the network can be an indication of a cyber attack. Unusual patterns or trends in the traffic can be detected by analyzing the communication between nodes of the network. One scenario for unusual patterns is high volume of data being transferred between two nodes. Some attacks that come under this category are flooding based DDoS attacks, exfiltration and ping flood attacks. The communication of a source IP with unusually high number of destinations also can be categorized as an anomaly. The source device in such cases is known as super spreader (Liu, 2013). All these vulnerabilities involve study of the communication patterns within nodes of the network.

Many vulnerability scanners identify vulnerabilities in a specific host. However, this identification in isolation is not very useful when a large network with many hosts is involved (Ammann, Wijesekera, & Kaushik, 2002). The effect of an attack on one host on the rest of the hosts in the network needs to be considered, in order to get an estimation of the damage done by an attack over the network. This analysis can be done using various graph concepts and theories, where the graph will be the large network under threat. Graph concepts such as reachability and degree of nodes can be used to analyze the graphs and find out the extent of damage of an attack on the network.

2.4 Community Detection

One of the most common topics related to large graphs is community detection. An informal definition of community structure is the organization of nodes in modules, such that the density of edges within a module is large compared to that between different modules (Fortunato, 2010). This feature of graphs is very widely used in disciplines where graphs are used as representation of the system.

Identifying communities has gained popularity for many reasons. Communities formed in large graphs divide the graph into fairly independent parts. Hence, working on individual independent parts makes the work easier and faster. For many cases where graphs are used for representation, the size of the graph gets too big to do computation for each node. In case of cyber networks, the graph consists of millions of nodes. The structure of such a network is usually examined at a high level, either at a level with nodes that control the movement of data, or groups of computers within which networking is high as compared to the networking with the rest of the network. Community detection also helps in visualizing dense networks. Efficient visualization of large networks makes analysis significantly easier. A clustering algorithm finds subgraphs, each of which is then replaced with a meta-node as a cluster. In this way, a coarse graph is obtained by replacing all subgraphs with their corresponding meta-nodes (Huang, X., & Huang, W., 2015). A graph that has less number of nodes and is representative of the original graph makes it visually more understandable.

2.5 Community Detection Algorithms

"A general approach for solving many large-scale graph problems, as well as most other classes of large-scale computational science problems, is through multilevel (multiscale, multiresolution, etc.) algorithms" (Ron, Safro, & Brandt, 2011, p407). Graph analytics works better if there is a coarse version of the graph, with communities of nodes, such that the communities form a coarse graph that represents the structure and properties of the original graph. Working on such graphs is faster than working on each host of the original network, especially for very large graphs with millions of nodes and edges. Community structure detection is considered to be a data analysis technique to explore features about the structure and behavior of a network. This section will shed some light on the various approaches taken for community detection in large networks.

2.5.1 Graph Metrics

While partitioning a graph into isolated subgraphs, the quality of the subgraphs needs to be considered. Various graph metrics have been used in previous studies, which quantify the quality of the partitions. This section will give a brief description of the various metrics that help in quantifying the quality of the partitions.

2.5.1.1 Edge Betweenness

One of the first metrics used for modern age community detection was edge betweenness. Vertex betweenness was studied and first proposed by Freeman (1977) as a measure of centrality of the vertex in the network. Girvan and Newman (2002) generalized Freeman's betweenness centrality to edges, which led to the introduction of edge betweenness. For an edge, it is defined as the number of shortest paths that run along it.

2.5.1.2 Modularity

Girvan and Newman (2004) defined a new metric, known as modularity. It was introduced as a stopping condition for the community detection algorithm that uses edge betweenness to calculate modules. Modularity is now used very widely for community detection and analysis of quality of partitions. This quantity measures the fraction of edges within a community less the expected value of the same fraction (Girvan & Newman, 2004).

$$Q = \sum_i (e_{ii} - a_i^2)$$

where $a_i = \sum_j e_{ij}$, which represents the fraction of links that connect to the nodes in community (Girvan & Newman, 2004).

"The modularity of a partition is a scalar value between -1 and 1" (Blondel, Guillaume, Lambiotte, & Lefebvre, 2008). According to Girvan and Newman (2004),

values of modularity for networks with good community structure fall within the range of 0.3 and 0.7.

2.5.1.3 Edge clustering coefficient

Edge-clustering coefficient was a local quantity used by Radicchi, Castellano, Cecconi, Loreto, and Parisi (2004) to make their algorithm computationally less intensive. It is the ratio of number of triangles or loops that the edge belongs to, to the number of loops that it might potentially be a part of. The main idea behind this calculation was that the nodes that are members of the loop would belong to the same community, as they are tightly knit to each other.

2.5.1.4 Similarity

Tan, Poletto, Guttag, and Kaashoek (2003) have considered a metric, which involves finding the similarity between hosts of an enterprise network in order to do the grouping to coarsen the graph. The grouping algorithm described in the paper classifies the nodes into groups depending on their connection habits. The partitioning is done based on the value of similarity between pairs of hosts. A high similarity value means that the 2 hosts have more chances of being grouped together. The main challenge is to calculate the similarity. For this, the term connection has been defined as a pair of hosts, which have a connection between them. A set of connections *C(h)* for a host h consists of all hosts which are connected to h (Tan et al., 2003). According to the paper

by Tan et al. (2003), once the connection set C is defined for all hosts, the similarity is defined as:

$$similarity(h_1, h_2) = |C(h_1) \cap C(h_2)|$$

Thus, similarity between 2 nodes is the number of neighbors common to both the nodes.

2.5.2 Algorithms

This section will shed some light on the various approaches taken for community detection in large networks. Each algorithm uses some metric to quantify the quality of the partitions made.

Radicchi et al. (2004) have divided the graph partitioning and community identification into two types: agglomerative and divisive. Agglomerative algorithms start with all nodes and no edges. Links are added iteratively depending on appropriate metrics in order to group nodes together into communities.

Girvan and Newman (2002) introduced one of the first modern age community detection algorithms. The Girvan Newman algorithm (GN algorithm) is a hierarchical divisive algorithm wherein communities are detected using “edge betweenness”. The edges that connect communities will have high edge betweenness. The algorithm involves calculating the edge betweenness for all edges and removing the edges with highest value of betweenness. The betweenness of the affected edges is recalculated. These two steps are repeated until no edges are left. The main disadvantage of this approach is the speed. This algorithm is very slow, with a worst case running time of

$O(m^2n)$, where m is the number of edges and n the number of vertices, or $O(n^3)$ for a sparse graph, where m is approximately equal to n. Another major disadvantage of this algorithm was that it provided no guidance on the optimal number of communities a network should be split into. (Newman, 2004) This means that this algorithm will make divisions in the graph even if the division makes no sense and isn't necessary.

In order to address the problems mentioned above, Newman (2004) introduced a new algorithm, which was much faster as compared to previous algorithms and gave excellent results on real networks. This algorithm is a hierarchical agglomerative algorithm, which uses modularity as the metric to quantify the quality of the partitions. The modularity is calculated as follows:

$$Q = \sum_i (e_{ii} - a_i^2)$$

where e_{ii} is the fraction of edges within module i and a_i is the fraction of edges with one end in module i (Newman, 2004). The algorithm starts with n communities, where n is the number of nodes. Initially, each vertex is the only member of one of the communities. In each step, communities are joined in pairs, depending on the gain in modularity. The communities that give maximum increase or minimum decrease in gain are chosen for the join. The gain in modularity is calculated by Newman (2004) using the following formula:

$$\Delta Q = e_{ij} + e_{ij} - 2a_i a_j = 2(e_{ij} - a_i a_j)$$

The worst-case running time of this algorithm is $O((m+n)n)$, or $O(n^2)$ on a sparse graph.

Clauset, Newman, and Moore (2004) performed the same greedy optimization of

modularity in their algorithm. However, they exploited some shortcuts for finding the optimization of modularity and used more sophisticated data structures to make the algorithm run faster. They used a heap tree of community pairs and a max heap of community pairs sorted by the modularity gain on merging of each community pair. Their algorithm has a running time of $O(n\log^2 n)$ (Clauset et al., 2004).

Blondel et al. (2008) have introduced another method to extract the community structure from large networks. They have used the Louvain method for community detection. The metric used to check the quality of the partitions is the modularity of the partition. Previous studies have introduced and optimized modularity for detecting communities in the network. Clauset et al. (2004) proposed the fastest approximation algorithm for optimization of modularity on large-scale networks. However, this method has a few drawbacks. It may produce significantly low values of modularity as compared to the values given by some other algorithms. Also, it has the tendency to group nodes together on networks that do not have a significant community structure. This algorithm addresses these drawbacks. This algorithm has been chosen for study and improvisation by the researcher. This method is explained in detail in the next section.

Duch and Arenas (2005) proposed a divisive algorithm that involves a different way of optimizing the modularity Q. Modularity is optimized using a heuristic search based on Extremal Optimization (EO) algorithm, which was proposed by Boettcher and Percus (2002). In extremal optimization, individual solution components are assigned a quality measure, known as “fitness”. In this modularity optimization problem by Duch &

Arenas (2005), the fitness measure will be related to the contribution of a node i to the calculation of Q, given a partition

$$Q_i = K_r(j) - k_i a_r(i)$$

where $K_r(i)$ is the number of links that i has with nodes within community r and k_i is the degree of node i. The contribution of node i to the modularity is given mathematically by:

$$\lambda_i = \frac{q_i}{k_i} = \left(\frac{K_r(i)}{k_i}\right) - a_r(i)$$

The value of λi is the fitness of node i (Duch & Arenas, 2005). The algorithm begins with the partition of the network into two subgraphs. The system then self-organizes by moving nodes with low fitness value to the other partition. The fitness value of affected nodes is recalculated. This is repeated till maximum value of modularity (Q) is reached. After this, all the edges between the two partitions are deleted. The same is done for each community to obtain multiple communities until the value of Q does not improve further. In comparison to the GN algorithm, this algorithm could dive deeper into the communities to find out communities that were difficult to reveal (Duch & Arenas, 2005).

There is some literature on community detection and graph coarsening in the cybersecurity field too. There have been approaches that have been taken to find modular structures in cyber networks for security applications.

Hogan, Johnson, and Halappanavar (2013) have used graph coarsening in order to find paths in the cybersecurity graph. A cybersecurity network can be represented as

a graph with nodes as vertices and the communication between them as edges. This study is narrowed down to a particular type of attack known as pass-the-hash attack. This attack targets single sign-on systems. When a user enters the password to log on, the hashed value of the password is hacked by the attacker. This hash value can then be used on other systems linked with the one to which the user logged on. The attacker does not need the cleartext password. Once the hacker gets access to the administrator, he can get more passwords, which will help him escalate the damage. This problem deals with reachability within the graph. One of the constraints while moving along the graph is that the credential in the credential store of the initiating machine must match a local administrator credential on the machine that is being logged into. “We define the reachability graph as the subgraph of the enterprise network topology created by starting with all nodes that have high value credentials (e.g. domain controller) and recursively adding edges to machines that meet these criteria” (Hogan, Johnson, et al., 2013). This reachability is computed by coarsening the graph. Traditional graph and matrix sparsification techniques speed up certain graph algorithms. However, these techniques might result in loss of some paths. Hence, the concept of graph minor has been used to coarsen the graph. A single edge is contracted at a time, depending on the degrees of the two vertices connected by this edge. This affects as few paths as possible. This form of graph coarsening gave a good approximation of the paths in the graph.

Hogan, Hui, Choudhury, Halappanavar, Oler and Joslyn (2013) have presented initial results of their research related to multiscale cybersecurity modeling. The research was with "the intent of achieving an efficient approximation of the system's state while trading off some level of precision in the process, with the intent of gaining an asymmetric advantage over a potential cyber attacker" (Hogan, Hui, et al., 2013). The concepts developed during this research will help in the efficient analysis of cyber systems for cyber security and defense. This study focuses on the mathematical concepts, without considering the application in the cyber space. A multiscale graph can be viewed as a traditional graph with hierarchical partitioning of its nodes. Each partition results in a set of vertices, which are grouped together as a supernode. After multiple levels of partitioning, the original graph is converted into a graph of supernodes. The edges between these supernodes are aggregate functions of the weights of all edges between the supernodes in consideration. This aggregate function returns a single value as the weight of the edge between the supernodes (min, max, mean, etc.). Hogan et al. (2013) have laid a foundation to use a multiscale graph approach to model cyber networks, for which, they have developed a multiscale version of the Floyd-Warshall algorithm. This version of the algorithm has two parts: one part calculates the diameter of the supernode while the second part calculates the array of minimum distances between supernodes. The time complexity improved from over $O(N^3)$ to $O(N^2)$, where N is the number of vertices in the original graph. The inference of

this study was that multiscale variants of basic algorithms could improve the efficiency significantly in exchange for loss in precision.

In this study, Hogan, Hui, et al. (2013) have also considered reachability in the multiscale scenario. Multiscale reachability is defined as follows: "Given two supernodes, π_i^j and π_i^k, are all nodes within π_i^k reachable from all nodes within π_i^j? If there is no path from π_i^j to π_i^k in G_i then clearly the answer is "no"" (Hogan, Hui, et al., 2013). However, the converse does not hold true. Connectivity between two supernodes does not mean that all subnodes of the supernodes are connected. A probabilistic approach towards reachability is discussed, in order to compute the probability that a subnode from π_i^j is connected to a subnode π_i^k, given that the two supernodes are connected.

2.6 The Louvain Method

This study focuses on using the Louvain method for large cyber networks. The Louvain method is an algorithm that carries out greedy optimization of modularity. The maximization of modularity is an NP complete problem (Brandes et al., 2006). Hence, several algorithms have been proposed in order to optimize modularity greedily in order to get good partitions in a reasonably fast way. This method is one of the best modularity optimization algorithms.

The Louvain method is an agglomerative algorithm that starts with each node assigned to a unique community. This algorithm runs multiple passes till the best partitions are achieved. Each pass further consists of two phases. In the first phase, for

each node i, the gain in modularity is calculated if the node is removed from its community and placed in the community of each of its neighbors. The modularity is calculated using the following formula:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \partial(c_i, c_j)$$

Here, A is the adjacency matrix representing the graph, $k_i = \sum_j A_{ij}$ is the total weight of links going in node i and $m = \frac{1}{2m} \sum_{i,j} A_{ij}$ is the total link weight over the whole network (Blondel et al., 2008). The equation used by Blondel et al. (2008) to calculate the gain in modularity is:

$$\Delta Q = \left[\frac{\Sigma in + k_{i,in}}{2m} - \left(\frac{(\Sigma tot + k_i)}{2m} \right)^2 \right] - \left[\frac{\Sigma in}{2m} - \left(\frac{\Sigma tot}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]$$

Here, Σin is the total link weight inside community *C*, Σtot is the sum of the link weights incident to community *C*, k_i is the sum of the weights of the links incident to node i, $k_{i,in}$ is the sum of the weights of the links from i to nodes in *C* and *m* is the sum of the weights of all the links in the network (Blondel et al., 2008).

The node i is removed from its community and placed in the community of the neighbor that gives maximum modularity gain, only if the gain is positive. In case there is no positive gain, i stays in its own community. The second phase of the pass is to create a new network whose nodes will be the communities calculated in the first phase. The weights between the new nodes are calculated by adding up all the weights between the nodes of the corresponding communities. The two phases are then carried out iteratively on the network created in the previous pass.

This algorithm is extremely fast due to the fact that the number of communities reduces drastically after the first few passes, making the amount of computations less in the later passes. Also, the gains in modularity are very easy to compute with the given formula. The time complexity of this algorithm is O(m), where m is the number of edges in the network.

2.7 Summary

This section gave a background about the study and related literature. This research involved studying the Louvain method and code thoroughly and trying to add heuristics to it to make it run faster.

CHAPTER 3. METHODOLOGY

This chapter gives details about the methodology and framework used in this study. This includes the hardware and software setup, the variables and the modifications carried out in the original algorithm.

3.1 Research Framework

The aim of this research was to study the Louvain method thoroughly and look for places where heuristics could be added to make the algorithm run faster, while preserving the accuracy at the same time. The results obtained were in the form of a statistical analysis, which brought about a comparison of the running time of the original algorithm and the modified algorithm proposed in this study.

3.2 Code Modifications and Heuristics added

This section includes a quick recap about the original Louvain algorithm and the heuristics and modifications added by the researcher in order to make the algorithm run faster.

The detailed implementation of the Louvain method is given in Section 2.6. The Louvain method starts with a partition where each node is in its own community. The

algorithm moves around and considers each node of the community in a random order. Each node is moved into its neighboring communities in order to look for an improvement in the modularity value (Q). ΔQ denotes the difference in modularity as a result of removal of node *i* from its community and insertion into a neighboring community *C*. Node *i* is inserted into the neighboring community that gives maximum gain in modularity. This is repeated until there is no further improvement in the modularity value. After this, a new graph is created using the formed communities and the same steps are followed till maximum gain in modularity is achieved.

The researcher started working on the algorithm by profiling the original implementation. In that process, it was observed that almost 40% of the time taken by the algorithm was spent on removing the node *i* from its community, placing it in every neighboring community and calculating the modularity gain. Hence, a heuristic was added that brought down the running time spent on moving around neighbors. Instead of considering all neighboring communities, the modified algorithm considers the community of the neighbor with the largest weight. This way, the algorithm does not visit every neighboring community and check the gain in modularity for that community.

The second change made by the researcher is the order in which the nodes are considered for modularity gain calculation. The original algorithm chooses each node randomly and carries out the removal, modularity gain calculation and insertion for that node. Instead of choosing nodes randomly, the researcher sorts the nodes according to the number of neighbors they have, starting with the node with least number of

neighbors. This heuristic is advantageous as it gives fixed results. In case of random choosing of nodes, the community structure and modularity values might change each time the algorithm is implemented. If the order is fixed, the community structure as well as modularity value is fixed.

The time complexity of the modified algorithm is O(n), where n is the number of nodes in the network. In the original algorithm, all nodes are considered randomly and for each node, all neighbors are considered. According to Matula (1987), the average degree of a node is 2m/n, where m is the number of edges and n the number of nodes.

The time complexity for the original algorithm is:

$$O(nk) = O\left(n \times \frac{2m}{n}\right) = O(m)$$

However, in the modified algorithm, for each node only one neighbor is considered, which is the neighbor with maximum weight. Hence the time complexity of the algorithm reduces from O(m) to O(n).

3.3 Drawbacks of the Modifications

One of the major drawbacks of the Louvain algorithm is resolution limit. Resolution limit is a phenomenon where communities that are smaller than a scale are not identified. It is a very common problem that is observed in most modularity optimization techniques. This study does not address this problem. Considering the order in which the nodes are considered for modularity calculation, the resolution limit will not reduce. However, as the main aim of this study is to speed up the community

detection process, and as the resolution limit problem is not affecting the quality of partition drastically, the researcher did not address this issue.

3.4 Experimental Setup

All the versions of the code have been implemented on a common hardware setup. This study used the Conte community cluster at Purdue University, provided by Information Technology at Purdue (ITaP) Research Computing (RCAC). Conte consists of 580 nodes and most nodes consist of identical hardware. The hardware specifications of the cluster are shown in Table 3.1.

Table 3.1: *Hardware specifications of Conte*

Specification	Value
Number of Nodes	580
Processors per node	Two 8-Core Intel Xeon-E5 + Two 60-Core Xeon Phi
Cores per node	16
Memory per node	64GB
Operating System	Red Hat Linux 6 (RHEL6)
Resource manager	TORQUE Resource Manager 4

3.5 Data Used

This study aimed to test the community detection algorithms on cyber networks for cyber security purposes. Hence, the data used by the researcher was a large dataset of IP traces in the form of packet captures (.pcap files). This section gives details about

the data used to test the algorithms and the cleaning and preprocessing needed before using it.

3.5.1 Source of Data

The data used in this research was an anonymized IP traces dataset obtained from the Center for Applied Internet Data Analysis (CAIDA). This dataset contains anonymized passive traffic traces from CAIDA's equinox-Chicago monitor on high-speed Internet backbone links for the year 2015. This study used only a chunk of the whole data. The data that was used was the traces that were collected on 29th February 2015. The traces in this dataset were anonymized for security issues. This was done using CryptoPan prefix-preserving anonymization. The anonymized traces are stored in pcap format and can be read using software like tcpdump, Wireshark, etc.

3.5.2 Data Format and Processing

The data was decoded and read using tcpdump. Tcpdump is a packet analyzing tool that allows the user to display packets being sent and received over a network. It is used to capture, view and analyze packets. As the data is already available, this study used tcpdump only to read and save the data in readable format. Running a tcpdump command gave details about the IP traces, including source IP address, destination IP address, source port number, destination port number, payload length and flag values in readable format. The data, after decryption is shown in Figure 3.1.

Figure 3.1. pcap data using tcpdump

The algorithm takes the source, destination and weight as input in the form of a text file or csv file. However, it was very inconvenient to feed in the IP addresses and use data structures for IP addresses. Hence, the IP addresses were converted to decimal values using the following conversion:

$$Decimal_{IP} = Octet1\times256^3 + Octet2\times256^2 + Octet3\times256^1 + Octet4\times256^0$$

Figure 3.2 shows a block diagram of the conversion mentioned above.

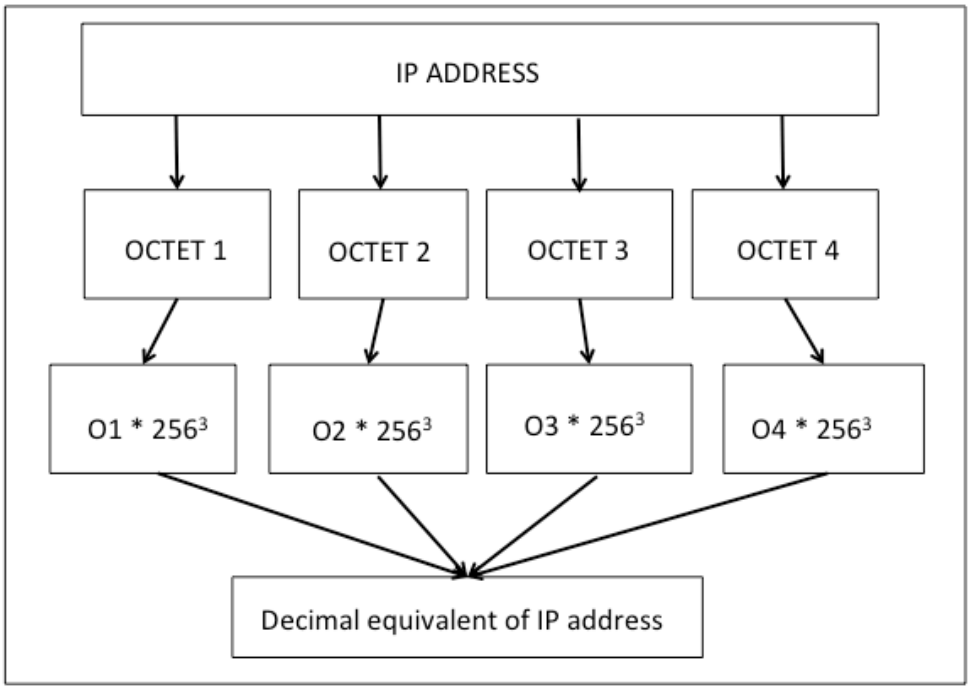

Figure 3.2. Conversion of IP address to decimal number.

3.6 Hypothesis

H_o: There is no improvement in the runtime of the modified Louvain algorithm proposed in this study as compared to the original algorithm.

H_a: There is an improvement in the runtime of the modified Louvain algorithm proposed in this study as compared to the original algorithm.

In statistical terms,

Ho: $\mu 1 - \mu 2 = 0$

Ha: $\mu 1 - \mu 2 < 0$

Where $\mu 1$ is the mean of the running times of the modified algorithm and $\mu 2$ is that of the original algorithm.

3.7 Variables

This section lists down the independent and dependent variables used to collect the results in this study.

3.7.1 Independent Variables

The performance of the algorithms in terms of running time will vary with the size of the network given as input, assuming that the hardware setup is constant. Thus, the researcher tried to observe how the algorithms perform for different network sizes. The sizes were varied in terms of nodes and edges. Hence, the independent variables used in this study were:

1. The algorithm used:

   The results were collected for the original algorithm and the modified algorithm proposed in this research.

2. The number of nodes:

   The original dataset consists of 3 million nodes. The researcher has taken different subsets of this data, with range from 0.5 million to 3 million nodes, with a step of 0.5 million.

3. The number of edges

   The original dataset consists of 6 million unique edges. The researcher has taken different subsets of this data, with range from 1 million to 6 million edges, with a step of 1 million.

3.7.2 Dependent Variables

The variables whose values are recorded and observed are mentioned in this section. They are as follows:

1. The running time

   The main property of the algorithms that is of concern in this study is the running time of the algorithms. The modularity value is monitored only to check if the quality of the partitions is preserved. Hence, the only dependent value is running time.

3.8 Summary

This section described the methodology, framework and variables used in this study

CHAPTER 4. RESULTS AND ANALYSIS

This chapter displays all the results collected after running the Louvain method and the modified Louvain method proposed in this study on different datasets. It also includes an analysis of the results.

4.1 Two sample t-test

The statistical test chosen for this study was a two-sample t test. It is commonly used to check if the difference between two groups is significantly different or not. A two-sample t test calculates a confidence interval and does a hypothesis test of the difference between two population means whose standard deviations are unknown. The samples of the two groups are collected independent of each other. The test was carried on various datasets with varying number of nodes and edges. Table 4.1 gives a list of the different sizes of the network that were considered.

Table 4.1 *Different sizes of network as input*

Parameter	Sizes					
Number of edges (millions)	1.0	2.0	3.0	4.0	5.0	6.0
Number of nodes (millions)	0.5	1.0	1.5	2.0	2.5	3.0

The researcher has chosen an alpha level of 0.05, which means that on repeating the experiment multiple times the results that are obtained have a 95% chance of being right. This level of significance was chosen as it was adequate for this study. As the experiments were simple, a lower level of significance was not needed. A significance level of 0.05 essentially means that if the p-value > 0.05 then the null hypothesis is not rejected and the results obtained are not statistically significant. If the p-value is greater than the significance level, the results obtained are statistically significant and the null hypothesis can be safely rejected. The hypothesis that was tested in this study was:

$H_o: \mu_1 - \mu_2 = 0$

$H_a: \mu_1 - \mu_2 < 0$

where μ_1 is the mean of the running times of the modified algorithm and μ_2 is that of the original algorithm.

4.2 Results for Varying Number of Edges

This section includes all the results obtained by running the original Louvain method and its modification on datasets having varying number of edges. The results are shown in the form of tables showing the statistical analysis as well as graphical representations of the results.

4.2.1 1 Million Edges

This dataset consists of 1,000,000 unique links. Both algorithms were run 100 times on this input. The observations that were recorded are summarized in Table 4.2.

Table 4.2 *Observations for 1 Million Edges*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time(s)	10.69	9.78
Modularity value	0.9565	0.9559
P value	<0.0001	

The p value shown in Table 4.2 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the researcher can say that the modified algorithm is faster than the original Louvain algorithm. Also, the average modularity values shown in the table qualify the quality partitions as good.

4.2.2 2 Million Edges

This dataset consists of 2,000,000 unique links. Table 4.3 shows the results collected for this input on both the algorithms.

Table 4.3 *Observations for 2 Million Edges*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	36.44	32.25
Modularity value	0.9262	0.9261
P value	<0.0001	

The p value shown in Table 4.3 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm. The modularity values for this input indicate that the quality of the partitions has been preserved.

4.2.3 3 Million Edges

This dataset consists of 3,000,000 unique links. Table 4.4 summarizes the results for this input. The average modularity values of the original algorithm is 0.931 and that of the modified algorithm is 0.931 as well, indicating exact preservation of the quality of quantities.

Table 4.4 *Observations for 3 Million Edges*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	45.37	41.46
Modularity value	0.931	0.931
P value	<0.0001	

The p value shown in Table 4.4 is below the significance, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm.

4.2.4 4 Million Edges

This dataset consists of 4,000,000 unique links. Table 4.5 includes the results for this input.

Table 4.5 *Observations for 4 Million Edges*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	108.43	79.94
Modularity value	0.9386	0.9179
P value	<0.0001	

The p value shown in Table 4.5 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm for 4 million edges. Also, the modularity values for this input are good.

4.2.5 5 Million Edges

This dataset consists of 5,000,000 unique links. The original algorithm gives an average modularity value of 0.9388, while the modified algorithm gives an average modularity value of 0.9376. Hence, the quality of the partitions is preserved. The details about the results are shown in Table 4.6.

Table 4.6 *Observations for 5 Million Edges*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	124.2	91.24
Modularity value	0.9388	0.9376
P value	<0.0001	

The p value shown in Table 4.6 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm.

4.2.6 6 Millions Edges

This dataset consists of 6,000,000 unique links. Both the original and the modified algorithm give an average modularity value of 0.9403. Hence, the quality of the partitions is preserved. The details about the running time of both algorithms are shown in Table 4.7.

Table 4.7 *Observations for 6 Million Edges*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	155.6	91.4
Modularity value	0.9403	0.9403
P value	<0.0001	

The p value shown in Table 4.7 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm.

4.2.7 Summary of results

This section gave the results obtained after running both the algorithms on varying number of edges. This was done in order to observe the change in performance of the algorithms with respect to the number of nodes. Figure 4.1 shows the

improvement of the modified algorithm using a scatterplot. Figure 4.2 represents the same results in the form of side-by-side boxplots. It is observed that the running time improves slightly for smaller number of edges and improves significantly for networks with larger number of edges.

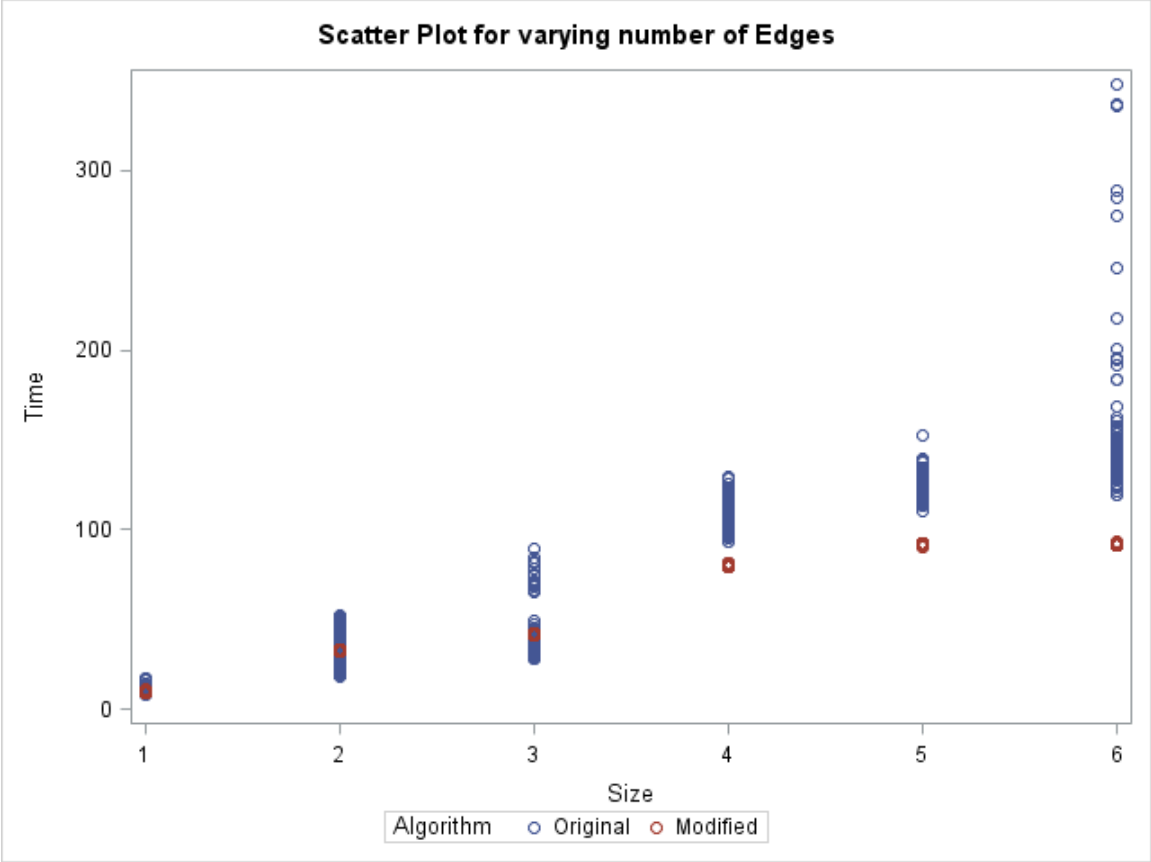

Figure 4.1 Scatterplot for varying number of edges

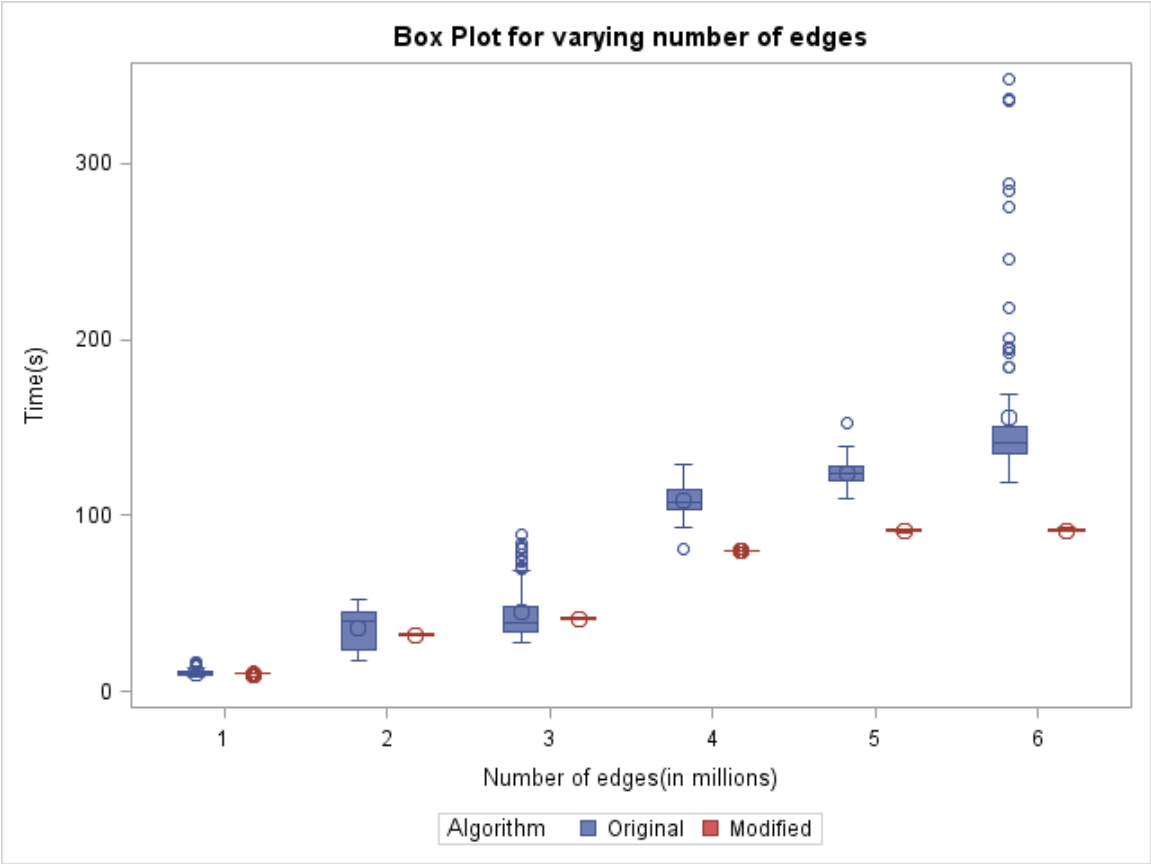

Figure 4.2 Boxplot for varying number of edges

4.3 Results for Varying Number of Nodes

This section includes all the results obtained by running the original Louvain method and its modification on datasets having varying number of nodes. The results are shown in the form of tables showing the statistical analysis as well as graphical representations of the results.

4.3.1 0.5 Million Nodes

This dataset consists of 0.5 million nodes. The details about the running time of both algorithms are shown in Table 4.8.

Table 4.8 *Observations for 0.5 Million Nodes*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	77.13	44.21
Modularity value	0.7852	0.783
P value	<0.0001	

The p value shown in Table 4.8 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm for 0.5 million nodes. The modularity values indicate that the quality of the partitions is preserved.

4.3.2 1 Million Nodes

This dataset consists of 1 million nodes. The original algorithm gives an average modularity value of 0.8148 while the modified algorithm gives an average modularity value of 0.8126. Hence, the quality of the partitions is preserved. The details about the results are shown in Table 4.9.

Table 4.9 *Observations for 1 Million Nodes*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	89.6	46.63
Modularity value	0.8148	0.8126
P value	<0.0001	

The p value shown in Table 4.9 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm for an input of 1 million nodes.

4.3.3 1.5 Million Nodes

This dataset consists of 1.5 million nodes. The details about the running time, modularity and the statistical results after running both algorithms are shown in Table 4.10. The modularity values for the original and modified algorithm are 0.8285 and 0.8267 respectively. This indicates the good quality of the partitions for both the algorithms.

Table 4.10 *Observations for 1.5 Million Nodes*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	99.67	57.82
Modularity value	0.8285	0.8267
P value	<0.0001	

The p value shown in Table 4.10 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm for this input.

4.3.4 2 Million Nodes

This dataset consists of 2 million nodes. The original algorithm gives an average modularity value of 0.8479 while the modified algorithm gives an average modularity value of 0.846. Hence, the quality of the partitions is preserved. The details about the performance of both algorithms are shown in Table 4.11.

Table 4.11 *Observations for 2 Million Nodes*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	110	57.79
Modularity value	0.8479	0.846
P value	<0.0001	

The p value shown in Table 4.11 is below the level of significance chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm for 2 million nodes.

4.3.5 2.5 Million Nodes

This dataset consists of 2.5 million nodes. Table 4.12 gives a summary of the results obtained on running both the algorithms with this dataset as input. The average values of modularity shown in the table are above 0.5, which is the threshold for the quality of partitions. Hence, this indicates that the quality of the partitions has been preserved.

Table 4.12 *Observations for 2.5 Million Nodes*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	117	61.1
Modularity value	0.8569	0.8547
P value	<0.0001	

The p value shown in Table 4.12 is below the significance level, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm.

4.3.6 3 Million Nodes

This dataset consists of 3 million nodes. The details about the running time of both algorithms are shown in table 4.13.

Table 4.13 *Observations for 3 Million Nodes*

Algorithm	Original Louvain	Modified Louvain
Number of runs	100	100
Average running time	129.1	67.78
Modularity value	0.8663	0.8645
P value	<0.0001	

The p value shown in Table 4.13 is below the significance level chosen for this study, which is 0.05, hence, the stated null hypothesis can be rejected. This leads to the inference that, with 95% confidence, the modified algorithm is faster than the original Louvain algorithm.

4.3.7 Summary of Results

This section described the performance of both the algorithms for different number of nodes. The modified algorithm worked much better than the original one in terms of running time. The modified algorithm is almost twice as fast as the original algorithm for each input. The scatterplot and side-by-side boxplot in Figures 4.3 and 4.4 respectively give a graphical representation of the running times.

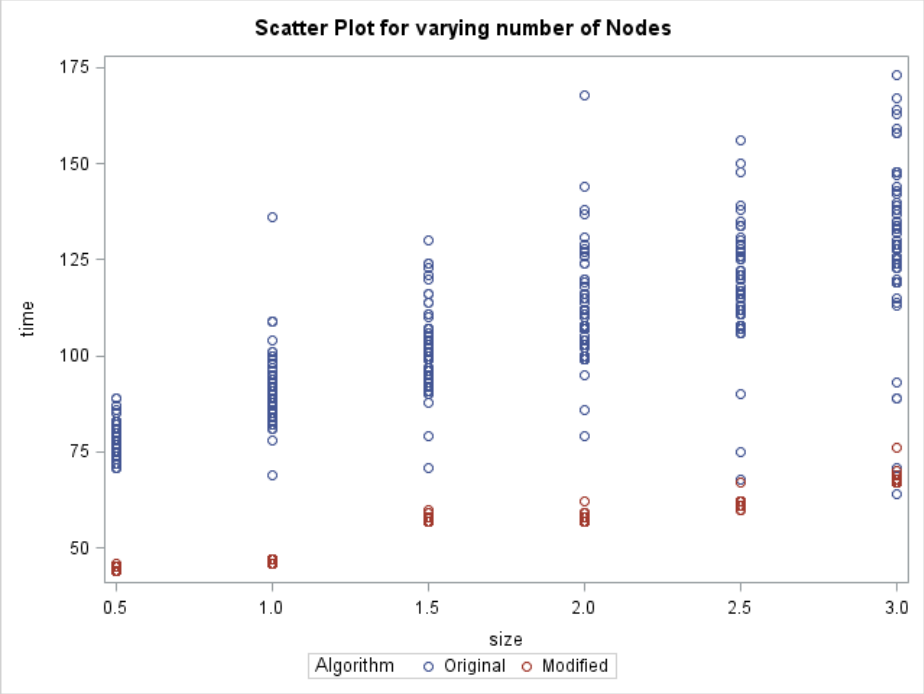

Figure 4.3 Scatterplot for varying number of nodes

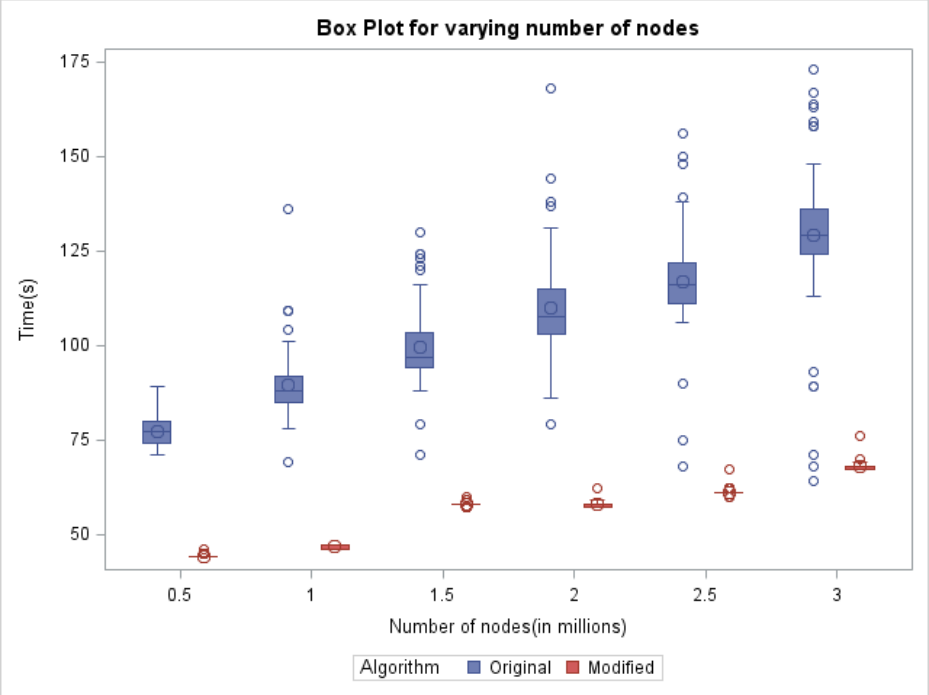

Figure 4.4 Boxplot for varying number of nodes

4.4 Overall summary of results

This section gives a brief summary of the results discussed in this chapter. Tables 4.14 and 4.15 show the average running times of both algorithms.

Table 4.14 *Average running times for varying number of edges*

Size of network	Average running time for Original Louvain method (s)	Average running time for Modified Louvain method (s)	Percentage reduction in running time (%)
1.0M Edges	10.69	9.78	8.51
2.0M Edges	36.44	10.43	11.5
3.0M Edges	45.37	41.46	8.62
4.0M Edges	108.43	79.94	26.28
5.0M Edges	124.2	91.24	26.54
6.0M Edges	154.18	91.4	41.26

Table 4.15 *Average running time for varying number of nodes*

Size of network	Average running time for Original Louvain method (s)	Average running time for Modified Louvain method (s)	Percentage reduction in running time (%)
0.5M Nodes	77.13	44.22	42.67
1.0M Nodes	89.6	46.63	47.96
1.5M Nodes	99.67	57.82	41.99
2.0M Nodes	110	57.69	47.55
2.5M Nodes	117	61.1	47.78
3.0M Nodes	129.1	67.68	47.58

Figure 4.5 shows a graph representing the data in Table 4.14. It is evident from the graph that the modified algorithm runs faster than the original algorithm. The difference is more significant for larger graphs.

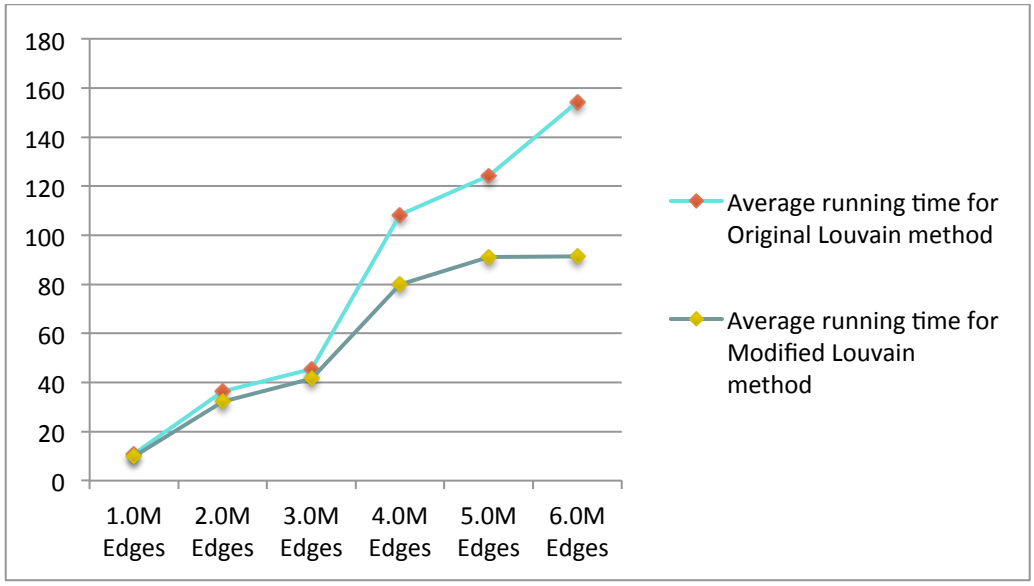

Figure 4.5 Summary of running time for varying number of edges

Similarly, Figure 4.6 shows graphically how the modified algorithm performs better than the original algorithm for varying number of nodes.

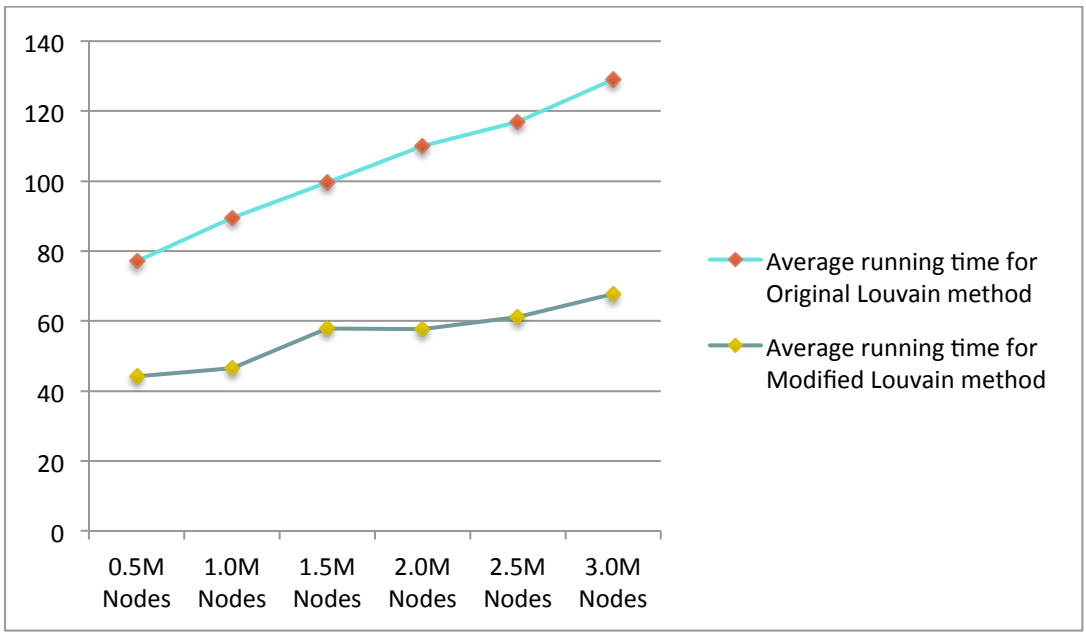

Figure 4.6 Summary of running time for varying number of nodes

4.5 Summary

This chapter presents the results of this study, including all tabular and graphical representation of the analyses. This chapter also does the hypothesis testing using the collected results.

CHAPTER 5. SUMMARY

This chapter is a final summary of the research conducted, the findings and the analysis of results. It also includes relevant discussions and future scope of this study.

5.1 Conclusions

This study was conducted with the motive of making cyber network analysis easier for cybersecurity. Community detection can be used to find a modular structure in the network and create communities, wherein each community will have nodes that are tightly connected with each other. Community detection to get a granular view of the network.

After carrying out a literature review on the various community detection algorithms and their performance on large networks, the Louvain method was chosen for further study, as it is one of the faster algorithms for finding a modular structure in large networks. Some heuristics were added to the algorithm to make it faster. At the same time, the modularity value was monitored to check if the quality of the partitions was preserved after making the modifications. Chapter 4 gives details about the results collected. The results showed that the addition of heuristics to the algorithm helped the algorithm speed up significantly.

5.2 Discussions

This study considered a serial version of the Louvain method for research. Along with adding heuristics to the serial algorithm, this study also looked into and proposed a parallel heuristic.

There are many scenarios to be considered and addressed while parallelizing the Louvain algorithm. They have been mentioned in a paper by Lu, Halappanavar, and Kalyanaraman (2015). One of them is the negative gain scenario. Considering two nodes *i*, belonging to C_1 and *j*, belonging to community C_2 are connected to any node in a community *C* and both of them decide to move into community C at the same time, the calculation of gain of modularity is affected, as two nodes are simultaneously trying to change the value of Q (Lu et al., 2015). A solution has been proposed in this study to address this problem. The algorithm below gives the steps for parallelizing the code using this parallelization heuristic.

Algorithm: Addressing the negative gain scenario and parallelizing the Louvain method. The inputs are a graph and an array of integers that represents the neighbors of each node (neighbor_id)

1. For each node n
2. For each neighbor j of n, set i^{th} bit of neighbor_id(n)
   - 1.1 If n=0
     - 1.1.1 validate=neighbor_id(n)
     - 1.1.2 Assign a thread to n
   - 1.2 Else
     - 1.2.1 If (validate &neighbor_id[n]) = 0
       Assign a thread to n
2. Implement modularity gain calculation for all threads simultaneously

Figure 5.1 Algorithm for parallelization of the Louvain method

5.3 Future Scope

This study has many directions in which further research can be conducted. Trying to parallelize the heuristics and adding new parallel heuristics to the code can make it faster. This research only focused on the running time of the algorithms. The quality of partitions can also be studied. The addition of heuristics will change the community structure. The extent to which the community structure has changed can be studied. It will be interesting to see if the addition of the heuristics gives better modules than those given by the original algorithm.

From the results collected, it was observed that the percentage reduction in the running time increased with an increase in the size of the network. The performance of the proposed algorithm can be tested on larger inputs to observe the extent of improvement for larger networks.

5.4 Summary

This chapter presents an overview of the results and a few thoughts on further improvement of the algorithm. It also considers the possible future work on this study.

REFERENCES

REFERENCES

Ahmed, M., Naser Mahmood, A., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, *60*, 19–31. http://doi.org/10.1016/j.jnca.2015.11.016

Ammann, P., Wijesekera, D., & Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (pp. 217–224). ACM.

Balaban, A. T. (1985). Applications of graph theory in chemistry. *Journal of Chemical Information and Computer Sciences*, *25*(3), 334–343. http://doi.org/10.1021/ci00047a033

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2008*(10), P10008.

Biener, C., Eling, M., & Wirfs, J. H. (2015). Insurability of cyber risk: An empirical analysis†. *The Geneva Papers on Risk and Insurance-Issues and Practice, 40(1), 131-158.*

Bliss, N. T., & Schmidt, M. C. (2013). Confronting the challenges of graphs and networks. *LINCOLN LABORATORY JOURNAL*, *20*(1).

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., & Wagner, D. (2006). Maximizing modularity is hard. *arXiv preprint physics/0608255*.

Duch, J., & Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Physical Review E*, *72*(2), 027104.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, *486*(3-5), 75–174. http://doi.org/10.1016/j.physrep.2009.11.002

Hogan, E., Hui, P., Choudhury, S., Halappanavar, M., Oler, K., & Joslyn, C. (2013). Towards a multiscale approach to cybersecurity modeling. In *Technologies for Homeland Security (HST), 2013 IEEE International Conference on* (pp. 80–85). IEEE.

Hogan, E., Johnson, J. R., & Halappanavar, M. (2013). Graph coarsening for path finding in cybersecurity graphs. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop* (p. 7). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=2459984

Huang, X., & Huang, W. (2015). GO: A cluster algorithm for graph visualization. *Journal of Visual Languages & Computing, 28, 71-82.*

Lu, H., Halappanavar, M., & Kalyanaraman, A. (2015). Parallel heuristics for scalable community detection. *Parallel Computing*, *47*, 19-37.

Matula, D. W. (1987, October). Determining edge connectivity in 0 (nm). In *Foundations of Computer Science, 1987., 28th Annual Symposium on* (pp. 249-251). IEEE.

Newman, M. E. (2003). The structure and function of complex networks. *SIAM review*, *45*(2), 167-256.

Newman, M. E., & Girvan, M. (2004). Finding and evaluating community structure in networks.*Physical review E*, *69*(2), 026113.

Park, S.-T., Khrabrov, A., Pennock, D. M., Lawrence, S., Giles, C. L., & Ungar, L. H. (2003). Static and dynamic analysis of the Internet's susceptibility to faults and attacks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies* (Vol. 3, pp. 2144–2154). IEEE.

Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., & Parisi, D. (2004). Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, *101*(9), 2658–2663.

Ron, D., Safro, I., & Brandt, A. (2011). Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, *9*(1), 407–423.

Tan, G., Poletto, M., Guttag, J. V., & Kaashoek, M. F. (2003). Role classification of hosts within enterprise networks based on connection patterns. In *USENIX Annual Technical Conference, General Track* (pp. 15–28).

Zihui Ge, D. R. F. (2001). On the hierarchical structure of the logical Internet graph. http://doi.org/10.1117/12.434397

APPENDIX

Table A *Results collected for 0.5 million nodes*

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
74	0.785172	45	0.783013
78	0.785268	44	0.783013
74	0.784037	45	0.783013
77	0.785206	44	0.783013
83	0.785055	44	0.783013
83	0.785252	44	0.783013
80	0.78515	44	0.783013
77	0.784653	44	0.783013
76	0.784236	44	0.783013
78	0.785076	44	0.783013
82	0.785414	45	0.783013
81	0.785136	44	0.783013
75	0.785376	44	0.783013
80	0.785089	44	0.783013
80	0.784911	45	0.783013
72	0.784501	44	0.783013
71	0.785282	44	0.783013
83	0.785039	44	0.783013
78	0.785366	45	0.783013
80	0.785118	44	0.783013
75	0.785437	44	0.783013
82	0.785424	45	0.783013
79	0.785328	44	0.783013
72	0.785227	44	0.783013
74	0.785269	45	0.783013
72	0.784295	44	0.783013
76	0.784689	44	0.783013
78	0.785565	45	0.783013
75	0.785406	46	0.783013
75	0.784374	44	0.783013
77	0.784994	44	0.783013
79	0.785197	45	0.783013
77	0.785284	44	0.783013
85	0.78515	44	0.783013

Table A (Continued)

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
80	0.785179	44	0.783013
79	0.785326	44	0.783013
81	0.785324	45	0.783013
83	0.784372	44	0.783013
77	0.785244	44	0.783013
75	0.785194	44	0.783013
77	0.784463	44	0.783013
72	0.784911	45	0.783013
77	0.784934	44	0.783013
83	0.78514	44	0.783013
76	0.785117	44	0.783013
75	0.785259	45	0.783013
89	0.785225	44	0.783013
78	0.785389	44	0.783013
71	0.785114	44	0.783013
89	0.785365	44	0.783013
74	0.785131	44	0.783013
78	0.784948	44	0.783013
74	0.785287	44	0.783013
83	0.785278	44	0.783013
74	0.785252	45	0.783013
78	0.785155	44	0.783013
75	0.785414	44	0.783013
77	0.78489	44	0.783013
77	0.785393	44	0.783013
75	0.785169	45	0.783013
75	0.785181	44	0.783013
72	0.784371	44	0.783013
78	0.785254	45	0.783013
74	0.785346	44	0.783013
78	0.78525	44	0.783013
71	0.785163	44	0.783013
80	0.785171	44	0.783013
74	0.785322	45	0.783013
78	0.785257	44	0.783013
83	0.785283	44	0.783013
87	0.785223	44	0.783013
75	0.785133	44	0.783013

Table A (Continued)

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
75	0.785204	44	0.783013
72	0.785356	44	0.783013
80	0.785085	44	0.783013
75	0.785307	44	0.783013
80	0.785269	44	0.783013
73	0.785143	45	0.783013
74	0.785102	44	0.783013
72	0.785292	44	0.783013
72	0.785118	44	0.783013
78	0.785488	44	0.783013
71	0.785008	44	0.783013
75	0.785104	44	0.783013
86	0.784533	44	0.783013
77	0.785032	45	0.783013
78	0.785275	44	0.783013
74	0.785228	44	0.783013
75	0.785009	44	0.783013
75	0.785042	44	0.783013
74	0.785163	44	0.783013
77	0.78508	44	0.783013
80	0.785317	44	0.783013
72	0.785342	44	0.783013
72	0.785458	44	0.783013
79	0.785289	44	0.783013
85	0.785029	44	0.783013
75	0.785215	44	0.783013
74	0.78512	45	0.783013
73	0.785371	44	0.783013

Table B: Results collected for 1 million nodes

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
87	0.814818	47	0.812592
87	0.814659	47	0.812592
88	0.815008	47	0.812592
82	0.815004	46	0.812592
84	0.814882	47	0.812592
82	0.81489	46	0.812592
94	0.815115	46	0.812592
99	0.814941	47	0.812592
87	0.815057	46	0.812592
94	0.815107	47	0.812592
91	0.814836	47	0.812592
109	0.815097	46	0.812592
92	0.814848	47	0.812592
88	0.814853	47	0.812592
87	0.81489	47	0.812592
81	0.814866	46	0.812592
85	0.814046	46	0.812592
95	0.814647	47	0.812592
85	0.814964	47	0.812592
82	0.814931	46	0.812592
92	0.814674	46	0.812592
84	0.8149	47	0.812592
90	0.814811	46	0.812592
97	0.815028	47	0.812592
94	0.815151	47	0.812592
82	0.814992	46	0.812592
85	0.815037	47	0.812592
101	0.814732	46	0.812592
85	0.814849	47	0.812592
98	0.81487	46	0.812592
136	0.814808	47	0.812592
91	0.814883	46	0.812592
92	0.8149	47	0.812592
88	0.814705	47	0.812592
91	0.813704	46	0.812592
85	0.814704	47	0.812592
88	0.814927	46	0.812592
88	0.814994	47	0.812592

Table B (Continued)

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
91	0.814881	46	0.812592
85	0.814831	47	0.812592
88	0.815001	46	0.812592
90	0.814963	47	0.812592
85	0.815078	47	0.812592
94	0.813988	46	0.812592
109	0.815077	47	0.812592
85	0.814856	47	0.812592
89	0.814941	46	0.812592
85	0.814943	47	0.812592
84	0.814485	47	0.812592
92	0.815085	47	0.812592
93	0.814635	47	0.812592
87	0.814613	47	0.812592
90	0.815017	46	0.812592
91	0.815095	47	0.812592
88	0.814108	47	0.812592
84	0.814789	46	0.812592
85	0.814899	47	0.812592
84	0.815042	47	0.812592
100	0.814901	47	0.812592
91	0.815065	46	0.812592
101	0.814575	47	0.812592
89	0.81486	47	0.812592
87	0.814609	47	0.812592
81	0.81517	46	0.812592
88	0.815031	47	0.812592
100	0.815004	47	0.812592
85	0.814231	46	0.812592
91	0.81405	47	0.812592
78	0.81491	46	0.812592
100	0.815109	47	0.812592
109	0.815069	46	0.812592
104	0.814911	47	0.812592
87	0.814198	47	0.812592
84	0.814791	47	0.812592
88	0.815085	46	0.812592
85	0.814048	47	0.812592

Table B (Continued)

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
85	0.815162	46	0.812592
85	0.814942	47	0.812592
85	0.814674	46	0.812592
96	0.815053	47	0.812592
69	0.814425	47	0.812592
87	0.814989	46	0.812592
90	0.814572	47	0.812592
85	0.814787	46	0.812592
91	0.814744	46	0.812592
94	0.814662	47	0.812592
93	0.814961	46	0.812592
82	0.81486	47	0.812592
87	0.814825	47	0.812592
90	0.81499	47	0.812592
85	0.814912	47	0.812592
93	0.814936	47	0.812592
84	0.814911	46	0.812592
88	0.81482	47	0.812592
98	0.814636	46	0.812592
86	0.81496	47	0.812592
81	0.814455	47	0.812592
87	0.814979	47	0.812592
83	0.814952	47	0.812592

Table C: Results collected for 1.5 million nodes

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
95	0.828403	58	0.826681
102	0.828598	58	0.826681
95	0.828088	58	0.826681
99	0.828393	59	0.826681
114	0.828277	58	0.826681
90	0.828337	58	0.826681
106	0.828577	58	0.826681
95	0.828124	58	0.826681
95	0.828533	57	0.826681
116	0.82845	58	0.826681
120	0.82841	60	0.826681
103	0.828631	58	0.826681
91	0.82847	57	0.826681
103	0.828786	58	0.826681
124	0.828383	58	0.826681
88	0.828254	58	0.826681
105	0.828601	58	0.826681
97	0.828373	57	0.826681
99	0.828661	58	0.826681
114	0.828515	58	0.826681
95	0.828413	58	0.826681
100	0.828412	57	0.826681
93	0.828587	58	0.826681
92	0.828641	58	0.826681
107	0.828617	57	0.826681
95	0.828441	58	0.826681
101	0.828526	58	0.826681
100	0.828687	58	0.826681
107	0.828431	57	0.826681
100	0.828655	58	0.826681
71	0.828346	58	0.826681
93	0.828656	58	0.826681
100	0.828547	57	0.826681
100	0.828765	58	0.826681
106	0.828601	57	0.826681
116	0.8284	58	0.826681
103	0.828646	58	0.826681
97	0.82864	57	0.826681

Table C (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
103	0.828729	58	0.826681
93	0.828557	58	0.826681
96	0.828354	58	0.826681
103	0.828451	57	0.826681
91	0.828465	58	0.826681
96	0.828632	58	0.826681
95	0.828377	59	0.826681
100	0.828627	58	0.826681
93	0.828596	58	0.826681
92	0.828633	58	0.826681
124	0.828541	57	0.826681
93	0.828298	58	0.826681
95	0.828201	57	0.826681
99	0.828632	58	0.826681
111	0.828686	58	0.826681
123	0.828715	57	0.826681
94	0.828482	58	0.826681
95	0.828617	58	0.826681
96	0.828184	58	0.826681
90	0.828421	58	0.826681
92	0.828691	58	0.826681
93	0.828447	58	0.826681
111	0.828416	58	0.826681
99	0.828502	58	0.826681
93	0.828539	58	0.826681
107	0.828383	58	0.826681
110	0.828492	58	0.826681
96	0.828491	58	0.826681
95	0.828456	58	0.826681
96	0.828602	57	0.826681
100	0.828541	58	0.826681
91	0.828347	58	0.826681
99	0.828721	58	0.826681
96	0.82843	58	0.826681
99	0.828432	58	0.826681
107	0.828557	58	0.826681
106	0.827639	57	0.826681
96	0.828705	58	0.826681

Table C (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
91	0.828647	58	0.826681
121	0.828547	57	0.826681
92	0.828506	58	0.826681
92	0.828173	58	0.826681
96	0.82834	58	0.826681
93	0.828486	58	0.826681
130	0.828561	57	0.826681
102	0.82792	58	0.826681
103	0.82869	58	0.826681
99	0.828554	58	0.826681
97	0.828535	58	0.826681
96	0.828652	57	0.826681
91	0.828394	58	0.826681
95	0.82867	58	0.826681
104	0.828579	58	0.826681
107	0.82818	58	0.826681
105	0.828551	57	0.826681
94	0.828572	58	0.826681
92	0.828583	57	0.826681
79	0.828272	58	0.826681
95	0.828668	58	0.826681

Table D: Results collected for 2 million nodes

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
100	0.847796	59	0.845967
103	0.847846	57	0.845967
103	0.84797	58	0.845967
112	0.847656	57	0.845967
107	0.847137	58	0.845967
107	0.847258	58	0.845967
100	0.847994	59	0.845967
116	0.848071	58	0.845967
102	0.847805	57	0.845967
103	0.847525	58	0.845967
108	0.848071	58	0.845967
112	0.848122	58	0.845967
108	0.848131	58	0.845967
102	0.84803	58	0.845967
103	0.847973	58	0.845967
99	0.848001	57	0.845967
112	0.847823	58	0.845967
86	0.847529	58	0.845967
104	0.848144	58	0.845967
115	0.84775	58	0.845967
131	0.848169	58	0.845967
103	0.848016	57	0.845967
99	0.847919	58	0.845967
120	0.847784	58	0.845967
104	0.84783	58	0.845967
107	0.84812	58	0.845967
103	0.84804	57	0.845967
115	0.847846	58	0.845967
137	0.847958	59	0.845967
99	0.848093	58	0.845967
107	0.847923	58	0.845967
103	0.847843	57	0.845967
116	0.847807	58	0.845967
107	0.848032	58	0.845967
111	0.848037	57	0.845967
116	0.847961	58	0.845967
99	0.848031	58	0.845967
112	0.848028	57	0.845967

Table D (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
108	0.847975	57	0.845967
119	0.848097	57	0.845967
128	0.847766	58	0.845967
124	0.847947	58	0.845967
127	0.848098	58	0.845967
116	0.84822	57	0.845967
120	0.847791	58	0.845967
111	0.847688	58	0.845967
107	0.847199	57	0.845967
112	0.847954	58	0.845967
100	0.848092	57	0.845967
115	0.847927	58	0.845967
107	0.848014	58	0.845967
107	0.848083	57	0.845967
111	0.847891	58	0.845967
103	0.847723	57	0.845967
107	0.847925	58	0.845967
168	0.847943	58	0.845967
126	0.848051	58	0.845967
110	0.847676	58	0.845967
107	0.847775	57	0.845967
99	0.848048	58	0.845967
103	0.847851	58	0.845967
115	0.847906	57	0.845967
126	0.847739	58	0.845967
112	0.847915	58	0.845967
114	0.848124	57	0.845967
79	0.847435	58	0.845967
103	0.848054	58	0.845967
100	0.848085	58	0.845967
108	0.847995	57	0.845967
107	0.84796	59	0.845967
124	0.848047	57	0.845967
108	0.847839	57	0.845967
127	0.847923	58	0.845967
100	0.847902	58	0.845967
104	0.84823	57	0.845967
104	0.847973	58	0.845967

Table D (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
108	0.848053	58	0.845967
108	0.847889	58	0.845967
103	0.84797	58	0.845967
105	0.84803	57	0.845967
107	0.848123	58	0.845967
103	0.848094	58	0.845967
103	0.847937	57	0.845967
138	0.848093	58	0.845967
99	0.847956	57	0.845967
129	0.847893	58	0.845967
100	0.848017	58	0.845967
112	0.847652	57	0.845967
118	0.848177	62	0.845967
115	0.847152	58	0.845967
111	0.847928	58	0.845967
144	0.847791	57	0.845967
95	0.848075	58	0.845967
112	0.847868	58	0.845967
103	0.847904	58	0.845967
99	0.848022	58	0.845967
100	0.84786	57	0.845967
108	0.847982	58	0.845967

Table E: Results collected for 2.5 million nodes

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
115	0.857124	62	0.854717
113	0.856747	61	0.854717
106	0.856992	61	0.854717
112	0.85686	61	0.854717
128	0.857056	61	0.854717
116	0.857	61	0.854717
135	0.85629	61	0.854717
111	0.856912	62	0.854717
116	0.856358	60	0.854717
127	0.856554	61	0.854717
112	0.857046	61	0.854717
117	0.856936	61	0.854717
138	0.857008	61	0.854717
116	0.856726	61	0.854717
139	0.856888	61	0.854717
125	0.856963	60	0.854717
120	0.857015	61	0.854717
134	0.856973	61	0.854717
112	0.857008	61	0.854717
150	0.856957	61	0.854717
156	0.856964	61	0.854717
131	0.857044	62	0.854717
90	0.856553	61	0.854717
126	0.856315	61	0.854717
111	0.856799	61	0.854717
111	0.857136	60	0.854717
134	0.856723	61	0.854717
134	0.857189	67	0.854717
126	0.856949	61	0.854717
106	0.856481	61	0.854717
117	0.856331	61	0.854717
107	0.856622	61	0.854717
68	0.856749	61	0.854717
107	0.856997	61	0.854717
130	0.856772	61	0.854717
122	0.85704	61	0.854717
112	0.856949	62	0.854717
108	0.856987	61	0.854717

Table E (Continued)

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
116	0.856805	61	0.854717
108	0.856871	61	0.854717
107	0.856404	61	0.854717
112	0.857029	61	0.854717
116	0.856601	61	0.854717
148	0.857241	61	0.854717
112	0.857086	61	0.854717
112	0.85687	61	0.854717
112	0.85707	61	0.854717
121	0.856859	61	0.854717
111	0.856236	61	0.854717
121	0.856782	61	0.854717
111	0.857	61	0.854717
125	0.857048	61	0.854717
116	0.856996	61	0.854717
115	0.856876	61	0.854717
113	0.856889	61	0.854717
129	0.857076	61	0.854717
117	0.857101	61	0.854717
117	0.856742	61	0.854717
117	0.856911	61	0.854717
108	0.85677	61	0.854717
115	0.856964	61	0.854717
120	0.857165	61	0.854717
121	0.856831	61	0.854717
107	0.856988	61	0.854717
108	0.856977	62	0.854717
112	0.85709	61	0.854717
75	0.856417	61	0.854717
122	0.856923	61	0.854717
112	0.857117	61	0.854717
108	0.856871	61	0.854717
121	0.857172	61	0.854717
134	0.857128	61	0.854717
107	0.856832	62	0.854717
108	0.856902	61	0.854717
113	0.856824	61	0.854717
108	0.856938	61	0.854717

Table E (Continued)

Original		Modified	
Time(s)	Modularity	Time(s)	Modularity
129	0.857004	61	0.854717
120	0.856993	61	0.854717
112	0.856907	61	0.854717
112	0.857042	61	0.854717
121	0.856898	61	0.854717
112	0.856196	61	0.854717
106	0.857017	61	0.854717
121	0.857017	61	0.854717
126	0.856964	62	0.854717
111	0.857261	61	0.854717
113	0.857122	61	0.854717
120	0.856282	61	0.854717
118	0.856863	61	0.854717
106	0.856256	61	0.854717
107	0.856908	61	0.854717
121	0.856298	61	0.854717
116	0.85683	61	0.854717
116	0.856768	61	0.854717
121	0.856953	61	0.854717
125	0.856812	61	0.854717
127	0.856999	61	0.854717
115	0.857062	61	0.854717

Table F: Results collected for 3 million nodes

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
129	0.866398	76	0.864477
129	0.866616	68	0.864477
147	0.866429	68	0.864477
129	0.866376	68	0.864477
125	0.866588	67	0.864477
113	0.866039	67	0.864477
128	0.865425	68	0.864477
163	0.866456	68	0.864477
128	0.866365	68	0.864477
167	0.866404	67	0.864477
124	0.866234	68	0.864477

Table F (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
129	0.8655	68	0.864477
124	0.866381	68	0.864477
129	0.86634	67	0.864477
129	0.866324	68	0.864477
119	0.86641	68	0.864477
148	0.866137	68	0.864477
128	0.866296	67	0.864477
124	0.866367	68	0.864477
173	0.865447	67	0.864477
133	0.866288	68	0.864477
120	0.866369	67	0.864477
124	0.865662	68	0.864477
119	0.866233	68	0.864477
125	0.866278	68	0.864477
133	0.866603	67	0.864477
64	0.866142	68	0.864477
158	0.866422	67	0.864477
164	0.866369	68	0.864477
137	0.86621	68	0.864477
89	0.865945	67	0.864477
144	0.866327	68	0.864477
158	0.866338	68	0.864477
133	0.866524	68	0.864477
128	0.866284	68	0.864477
125	0.865673	67	0.864477
124	0.866328	68	0.864477
135	0.866353	67	0.864477
128	0.866336	68	0.864477
132	0.866372	68	0.864477
123	0.866268	68	0.864477
125	0.866271	67	0.864477
134	0.866212	68	0.864477
130	0.866117	67	0.864477
143	0.866301	68	0.864477
119	0.86636	68	0.864477
114	0.866489	70	0.864477
138	0.86651	68	0.864477
126	0.866152	68	0.864477

Table F (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
139	0.866517	67	0.864477
119	0.866228	68	0.864477
129	0.866278	68	0.864477
159	0.866608	67	0.864477
129	0.865888	68	0.864477
129	0.866276	68	0.864477
119	0.866274	68	0.864477
124	0.866486	67	0.864477
135	0.866443	68	0.864477
148	0.866443	68	0.864477
124	0.866154	68	0.864477
124	0.866528	67	0.864477
129	0.865732	68	0.864477
129	0.866187	68	0.864477
138	0.866454	67	0.864477
124	0.866357	68	0.864477
129	0.866347	68	0.864477
129	0.866114	67	0.864477
129	0.86624	68	0.864477
133	0.866366	67	0.864477
138	0.866306	68	0.864477
123	0.866458	68	0.864477
68	0.865422	67	0.864477
129	0.866441	68	0.864477
158	0.866479	68	0.864477
123	0.866185	67	0.864477
124	0.866352	68	0.864477
139	0.866393	67	0.864477
129	0.866374	67	0.864477
129	0.865721	68	0.864477
119	0.86633	68	0.864477
148	0.866428	67	0.864477
143	0.866448	69	0.864477
142	0.865852	67	0.864477
71	0.866119	68	0.864477
128	0.866279	67	0.864477
89	0.866039	67	0.864477
148	0.866254	68	0.864477

Table F (Continued)

Original		Modified	
Time(s)	Modularity	time(s)	Modularity
134	0.866352	67	0.864477
128	0.86648	68	0.864477
124	0.866258	68	0.864477
134	0.865735	68	0.864477
93	0.866169	68	0.864477
123	0.866558	67	0.864477
124	0.866305	68	0.864477
115	0.866324	67	0.864477
140	0.86622	68	0.864477
119	0.86629	68	0.864477
129	0.866545	68	0.864477
138	0.866223	68	0.864477