Aalto University
School of Science
Department of Information and Computer Science

Henok Alene

Graph Based Clustering for Anomaly Detection in IP Networks

Master's Thesis
Espoo, October 25, 2011

Supervisors: Professor Erkki Oja, Aalto University

Instructor: PhD. Kimmo Hätönen, Nokia Siemens Networks

Aalto University
School of Science

Aalto University
School of Science

Aalto University
School of Science
Degree Programme of Information and Computer Science

ABSTRACT OF
MASTER'S THESIS

Author:	Henok Alene
Title of thesis:	Graph Based Clustering for Anomaly Detection in IP Networks
Date:	October 25, 2011 **Pages:** 7 + 67
Professorship:	Information and Computer Science **Code:** T-61
Supervisors:	Professor Erkki Oja
Instructor:	PhD. Kimmo Hätönen

In IP networks, an anomaly detection system identifies attacks, device failures or other unknown processes that deviate from the normal behavior of the network known as anomalies. The thesis studied anomaly detection in traffic datasets from IP networks. The datasets contained high number of normal events and few anomalies. This resembles a normally operating network.

We construct graphs from traffic data and study their properties. We formulated anomaly detection as a graph based clustering problem. A novel graph bi-partitioning algorithm called `NodeClustering` was designed to separate normal samples from anomalous ones.

Performance of `NodeClustering` was investigated with extensive network traffic data. The performance was compared with state of the art graph based spectral clustering algorithms. `NodeClustering` identified all the known intrusions in the data and outperformed the compared graph based methods with an average improvement of 50% on the true positive rate with lowest false positive rate on the studied datasets. In addition, its applicability to one non-traffic dataset was shown.

`NodeClustering` can be used in IP networks to detect anomalies. In the future, threshold used for graph partitioning can be studied further and computationally efficient methods to construct larger graphs might be studied.

Keywords:	Anomaly, Anomaly detection, Graphs, Node, Node degree `NodeClustering`,Weight matrix, partition, Clustering
Language:	English

Acknowledgements

I would like to thank my instructors Kimmo Hätönen, Perttu Halonen and supervisor Professor Erkki Oja for their valuable advices and encouraging comments. I would like to thank also staff members in Nokia Siemens Networks for enlightening discussions during the thesis work. I would like to thank my family for supporting me through out my studies and all in my life. I would like to give special thanks to my girlfriend Betelhem, who encouraged me to work hard and focus well during my studies and in my thesis.

The thesis was done in Nokia Siemens Networks (NSN) Research, Security Technologies located in Espoo in a framework of Finnish Strategic Center for Science, Technology and Innovation in the field of ICT (TIVIT's) Future Internet (FI) program. FI aims to improve practices and methods of the current Internet and transform it to the future. These include creating new methods, technologies, platforms, topologies or concepts to improve the current protocols, algorithms in security, privacy, accessibility, congestion, routing, and other aspects. This work was supported by TEKES as part of the Future Internet program of TIVIT.

Espoo October 25, 2011

Henok Alene

Abbreviations and Acronyms

ART	Adaptive Resonance Theory
DARPA	Defense Advanced Research Project Agency
FSM	Finite State Machine
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
NBA	Network Behavior Analysis
NSN	Nokia Siemens Networks
pcap	packet capture
RBF	Radial Basis Function
SOM	Self Organizing Map
SSH	Secure Shell
RIP	Routing Information Protocol
RPC	Remote Procedure Call
RTP	Real-Time Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UML	Unified Modeling Language
UDP	User Datagram Protocol

Contents

List of Tables

List of Figures

List of Symbols

$C(v_i)$	Centrality of node v_i
d	dimensionality of data vectors
d_i	Degree of node i
$\mathbf{d}$	matrix containing all d_i 's
$\mathbf{D}$	Node degree matrix containing d_i along its diagonal
$\mathbf{E}$	Edges of a graph
$e_{i,j}$	edge strength between node i and j
$\mathbf{G}$	Graph (directed or undirected)
$\mathbf{J}$	Objective function
$\mathbf{L}$	Un-normalized Laplacian matrix
$\mathbf{L_n}$	Normalized Laplacian matrix
l	eigenvector
$p_{i,j}$	probability of transition from node i to node j
$\mathbf{P}$	Transition Matrix
$\mathbf{P_d}$	Node degree distribution
$\mathbf{P}(i,\cdot)$	set of transition probabilities to all nodes except i^{th} node
S_v	Sub-vertices of the graph
$\mathbf{V}$	Vertices of a graph
v_i	i-th vertex of a graph
$\mathbf{W}$	Adjacency matrix
$w_{i,j}$	i^{th} row an j^{th}column entry of $\mathbf{W}$
$\mathbf{X}$	Traffic data containing the flows
$\mathbf{X_N}$	Normalized traffic data of the flows
y_i	disparity of a node
$\mathbf{Y}$	Disparity Matrix
λ	eigenvalues
σ	scaling parameter
$\boldsymbol{\tau}$	threshold of bi-partition

List of Algorithms

Chapter 1

Introduction

1.1 Scope of the thesis

Different components and services in a network generate traffic data which exhibit different patterns. Most of these patterns represent normal behavior. However, there are few cases that deviate from the normal pattern. The cause for the deviations might be due to malfunctioning device (for example a router or switch), attacks (intrusions) to the network or other unknown activities with abnormal patterns. We call such patterns anomalous patterns or anomalies.

The main focus of the thesis is finding anomalous samples in data sets collected from a normally operating Internet Protocol (IP) network environment. In a normal network environment, majority of the traffic samples are free of anomalies and few contain anomalies. One reason is users in the network usually perform normal activities, for instance web browsing, that do not generate malicious traffic. Moreover, there are different mechanisms that assure the normality of the network. Examples include anti-viruses and firewalls. The specific amounts of normal and anomalous samples in the traffic data from such environments vary depending on the environment itself where the log data is collected.

Network security analysts and system administrators usually need to find anomalies in a log data collected from some monitored network. In this thesis, we provide a novel algorithm for anomaly detection in a log data collected from IP network traffic.

1.2 Contribution

In machine learning, graph based data analysis has been studied very well. In this thesis, we represent log data from IP network data as a graph and formulate anomaly detection as a graph based clustering problem.

In this thesis, a new graph based clustering algorithm called `NodeClustering` is introduced. This algorithm is capable of bi-partitioning non-similar nodes of the graph into non-intersecting sets using a heuristic threshold. It is computationally inexpensive compared to currently existing graph based spectral clustering algorithms and outperforms them in the detection results as well.

To our best knowledge, this is the first method that does clustering on the nodes of the graph without utilizing the eigenvectors of the graph Laplacian for anomaly detection purposes. We test the performance of our algorithm with two traffic datasets.

1.3 Structure of the thesis

In the previous sections, we presented the thesis problem and our original contribution to the problem. The rest of the thesis follows the following structure.

The thesis proceeds in Chapter 2 with a general introduction of intrusion and anomaly detection systems.

Chapter 3 discusses the use of graphs in data analysis and different properties of graphs useful for the analysis.

Chapter 4 introduces graph based clustering methods and propose a new algorithm called `NodeClustering`.

In Chapter 5, we do anomaly detection with graphs on real traffic datasets

and make comparisons with other graph based spectral algorithm methods.

Finally, Chapter 6 concludes the thesis with the key outcomes of the experimental findings and makes recommendations for future work.

Chapter 2

Intrusion and Anomaly Detection Systems in IP Networks

In this chapter, an overview of different intrusion detection systems with emphasis on anomaly detection systems will be given.

2.1 Intrusion Detection Systems

In network security, an intrusion is defined as unauthorized action on a single host or a network to access data, use or modify the data and make the network unreliable and unsecured [1].

Intrusion Detection System (IDS) defines a set of hardware or software systems that automate the process of monitoring different activities in a single host or in networks to detect intrusions or indication of intrusions from collected log data or a live network traffic [2]. Deploying an IDS in an organization network or in a single host helps increasing the reliability and security of the network by detecting potential breaches of security [3].

We follow the process model of intrusion detection system [3], which describes IDSes with three elements. The elements are: different information sources generating traffic data, analysis methods for the data and response after the analysis [3]. We briefly describe each of the elements below.

2.1.1 Information sources

Information sources define different origins of incident information in the telecommunication network. These sources provide the data which can be used to detect whether intrusions have occurred [3]. The sources can send either packets from their networks and other elements or generate data from their internal operating systems or applications running on them.

They can further be divided into three types: network-based, host-based and application-based IDSes.

Network-Based IDS

A network-based IDS acquires and examines network traffic packets for signs of intrusions. A network-based IDS comprises a set of dedicated sensors or hosts which scan network traffic data to detect attacks or intrusive behaviors and protects the hosts connected to the network [3].

The major advantages of a network-based IDS include its ability to scan large networks, transparency to the normal operation of the network, ability to scan the traffic passively without being visible and that it can be made invisible to attackers and thus made more secure [3].

The major disadvantages of a network-based IDS are inability to handle encrypted data, incapacity to report whether an attack was successful or not but report only the initiation of an attack, and incapability to handle fragmented packets which make the IDS unstable [3].

Host-Based IDS

A host-based IDS operates on data collected from a single computer system (host). These data can be from the innermost part of the host's operating system (audit data) [1] or system log data. Host-based IDS uses these data to detect traces of an attack [3].

The advantages of host-based IDSes include the ability to detect incidents local to the host which might not be detected by the network-based IDS and

capacity to find attacks which target the host's operating system [3].

Host-based IDSes have some problems. They are usually deployed in the host system and can be disabled as part of the attack. Moreover, they use the host's computational infrastructure which makes its performance degrade. This type of IDS is deployed in individual hosts that make the configuration difficult as the different hosts have different behaviors and usage [3].

Application-Based IDS

Application-based IDS finds attacks originating from software applications running on a host. This IDS runs on a host and can be considered an element of the host-based IDS [3].

Application-based IDSes are advantageous since they keep track of the intercommunication between the application and the user. One major challenge of this IDS is that software application data can easily be modified and used as part of the attack [3].

2.1.2 Techniques of Analysis

In subsection 2.1.1, we have introduced different sources of network traffic data and where the IDSes can be placed. There are two major techniques IDSes use in analyzing the network traffic data: misuse and anomaly detectors.

Misuse Detection

Misuse detection systems encode a known attack in some pattern or signature. These detectors find patterns similar to the signature and flag them as attacks [4]. The signatures usually include patterns for different variants of the known attack, and non-attack behavior.

The main challenges of a misuse detection system are its inability to write signature models for all the known intrusive patterns and identifying which alike pattern is not intrusive. A misuse detection system fails when there

are attacks with new observed patterns that do not have previous recorded signatures [4].

Anomaly Detection

The term anomaly defines an activity whose behavior deviates from the normal pattern or profile. Anomaly detection systems define a model that detect anomalies. These systems define attacks as subset of anomalies [4].

Anomaly detection systems model normal activity patterns and report profiles that deviate from the normal pattern as anomalies.
Thus, anomalous behavior that is not intrusive can be reported as intrusive or anomalous sample can be reported as normal. The former case results in false-positive rates, which is a challenge in anomaly detection systems [4].

In anomaly detection systems, categorizing an anomaly that is intrusive in nature as a non-intrusive is a worse behavior than detecting an anomaly that is non-intrusive as intrusive.

2.1.3 Response

There are different measures the IDS take to combat the detected attacks or other anomalies. These responses can be either active, passive or a combination of both [3].

Active Response

Active response describes a set of automated actions taken when an attack is detected. The responses can be immediate obstructing of any access from the attacker when there is sign of the attack.

An attacker can be denied access by offensive measures (attacking back the intruder), collecting additional data to help examine the nature of the attack, or diagnose the attack and eventually catch the attacker [3].

Passive Response

Passive responses define a set of measures that a human agent, for example a system administrator or a security analyst, takes care of after receiving a report from the detection system. Some IDSes display the report as an alarm, which is visible through a pop-up window or some other visualization methods. The alarm can be a notification of an incident has occurred or exhibiting a more detailed description of the incident. Other IDSes send the report to a central management console through Simple Network Management Protocol (SNMP) traps and messages [3].

2.2 Anomaly Detection Systems

Anomaly detection systems are one technique used in intrusion detection systems. These systems construct a model which depicts normal behavior of a communication network. We discuss what the detection models use as input and how the models achieve detection focusing on three detection modeling approaches.

Some of the models in anomaly detection systems use raw data features as input while other models use preprocessed features. Features can be packet information (for example, packet count, duration of the packet, number of sent and received packets), used communication protocols (for instance, Transmission Control Protocol, User Datagram Protocol), generic behavior of normal users and other information.

The detection models use different approaches. In the following subsections, we discuss three modeling techniques used in anomaly detection systems [5].

2.2.1 Statistical-based models

Statistical-based anomaly detection models create a representation of the network normal behavior using different features of the network traffic data. When new pattern is observed, scores of anomaly are reported after the comparison with the normal model [5].

The normal models can be univariate which model only one feature, or mul-

tivariate which use many features of the traffic data [5].

There are some problems with statistical-based anomaly detection models. One problem is that the normal model can be trained mistakenly with an attack traffic instead of normal traffic. Selecting which features the models should use is another challenge [5].

2.2.2 Knowledge-based models

Knowledge-based anomaly detection techniques build a model using an acquired prior knowledge from an expert. The knowledge can be represented in form of rules and descriptions. Some well known knowledge based techniques include Finite State Machines (FSM), description languages like N-grammars and Unified Modeling Language (UML), and expert-systems [5].

The main advantage of this type of models is their flexibility to handle different anomalies in the specified knowledge. The problem with knowledge-based techniques is the lack of first-rate knowledge when making the rules and specifications [5].

2.2.3 Machine learning-based models

Machine learning-based techniques provide models adaptable with the data. When a new pattern is learned from data, the models adapt to incorporate this pattern [5]. This makes this method robust to detect anomalies than the previous models.

Examples of machine-learning based models include neural networks, Markov models, fuzzy logics, genetic algorithms, and clustering methods [5]. Although these models provide accurate identification results, building the models might be computationally expensive [5].

Chapter 3

Graphs in Data Analysis

In this Chapter, we discuss the application of graphs for data analysis, how to build graphs from data, and study different properties of graph.

3.1 Definition of a graph

A graph is a mathematical representation of a form $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \{v_1, v_2, \cdots, v_n\}$ represents the set of vertices (nodes) and $\mathbf{E} = \{e_{i,j}\}$ denotes a set containing individual edge values (strengths) $e_{i,j}$ connecting the vertices $v_i, v_j \in \mathbf{V}$ [6].

The number of vertices of the graph is $|V| = n$, where n is the size of the set $\mathbf{V}$. The nodes of a graph v_i and v_j are called neighbors if there is an edge $e_{i,j}$ between both nodes. This neighborhood relation can be written as $v_i \sim v_j$ [8]. A graph can be weighted or unweighted depending on the edge values $e_{i,j}$ between the nodes v_i and v_j.

A graph $\mathbf{G}$ is said to be *weighted* if the edge value $e_{i,j} \geq 0$, that is the edges have non-negative values [6]. $\mathbf{G}$ is called *unweighted* if $e_{i,j} = 1$ when $v_i \sim v_j$ and 0 otherwise [8].

$\mathbf{G}$ is called *directed* iff $e_{i,j} \neq e_{j,i}$, that is if the edge values in $\mathbf{E}$ are not symmetrical. $\mathbf{G}$ is said to be *undirected* if $e_{i,j} = e_{j,i}$, in other words $\mathbf{E}$ is symmetrical for any node v_i and v_j.

3.2 Data analysis with graphs

Graphs can represent different systems and their interactions. Some examples that can be modeled with graphs include biological networks, telecommunication networks, social interactions, a cloud computing environment, and operating system calls.

A graph can represent some or all entities of a systems. In a biological network, a graph might represent the interaction between different genes where the nodes represent genes and the edges might represent how one gene (node) affects the others during some metabolism reaction. In an operating system call, a graph can represent different system calls with its nodes and the order of the calls with the edge values. In social sciences, a graph might model the different people in the network as nodes and their degree of friendship or degree of interaction with its edge values.

In general, one can represent a given data with graphs. The node of the graph will represent each observation the data, thus the number of nodes will be equivalent to the number of observations. The edges of the graph can represent the similarity of each of the nodes.

If matrix $\mathbf{X}$,

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & . & . & x_{1,d} \\ . & . & & . \\ . & & . & . \\ . & & . & . \\ x_{n,1} & . & . & x_{n,d} \end{bmatrix}$$

represents the data where n defines the number of observations and d represents the number of variables (features) of the data, then the graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ will represent each observation of the data with each $v_i \in \mathbf{V}$ and their similarity with $e_{i,j} \in \mathbf{E}$.

The graph $\mathbf{G}$ will capture the similarity of the samples if a suitable function is used for computing the similarities. Depending on what the data $\mathbf{X}$ represent, $\mathbf{G}$ can be directed, undirected, weighted or unweighted.

If observations in the data have information indicating which nodes affect

behavior of other nodes, the graph will be directed. The behavior can be some cause-effect relation between the nodes. If no such information is available, the graph becomes undirected.

If there is a discrete information indicating which nodes in the graph co-occur with other nodes, the graph will be unweighted. The information can be given with some indicator function with $e_{i,j}$ 0 or 1, or a binary function indicating neighborhood relationship of the nodes. If there is numerical value that is not discrete for the edges where $e_{i,j} \geq 0$, the graph is weighted.

The following section presents how to construct weighted graphs and different types of weighted graphs.

3.3 Graph construction

3.3.1 Similarity functions

Similarity functions calculate the similarities between observations in the data. For instance, pairwise distances among samples of the data (which are nodes of the graph) can represent the similarity among the nodes. These functions mainly affect how many samples can be used to construct the graph and usually use distance functions to compute the similarities. Some examples of distance functions are Euclidean distance, cosine distance, and Jaccard distance.

The choice of distance methods is usually motivated by the task. In text and document processing and grouping tasks, cosine distance is commonly used. Jaccard distance is used in computing distance for binary attributes (features). Euclidean distance is the most used method for most machine learning tasks.

Distances are usually calculated in the original input space. There has been research that has shown that transforming from the input space to a higher dimensional space captures more interesting properties of the data. Kernels are mathematical functions that can achieve this transformation.

An input space is where the original samples are located. For the dataset $\mathbf{X}$, the input space is $\mathbb{R}^d$. A kernel function $\mathbf{K}(\mathbf{x_i}, \mathbf{x_j})$ transforms the similarity values of $\mathbf{x_i}$ and $\mathbf{x_j}$ from the input space to $(n \times n)$ kernel matrix.

Table 3.1 [10] makes a brief list of the kernels. Radial Basis Function (RBF) kernel, which uses the Gaussian kernel [11] with width or scaling hyper-parameter, is the most popular one. Selecting the hyper-parameters is a choice of modeling and can be motivated by prior expert knowledge or using methods like cross-validation. There are also other types of kernels, for example linear, polynomial, diffusion (heat), exponential, Cauchy, hyperbolic tangent, etc. Some of these kernels also have hyper-parameters, which can be chosen similar to the RBF kernel.

Types of kernels, $\mathbf{K}(x_i, x_j)$		
function	**Definition**	**hyper-parameter(s)**
RBF	$exp(-\frac{\|x_i - x_j\|^2}{\delta^2})$	$\delta > 0$
Exponential	$exp(-\frac{\|x_i - x_j\|}{\delta})$	$\delta > 0$
Sigmoid	$tanh(\alpha \langle x_i, x_j \rangle + \beta)$	$\alpha > 0, \beta < 0$
Logarithmic	$-log(1 + \|x_i - x_j\|)$	-

Table 3.1: Types of kernels

The kernels will provide a square matrix $\mathbf{W}$, where $\mathbf{W} = \mathbf{K}(\mathbf{x_i}, \mathbf{x_j})$, also called the adjacency matrix. Each entry $w_{i,j} \in \mathbf{W}$ captures pairwise similarities between sample i and j. The adjacency matrix is a full $n \times n$ matrix.

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & . & . & w_{1,n} \\ . & . & & . \\ . & & . & . \\ . & & . & . \\ w_{n,1} & . & . & w_{n,n} \end{bmatrix}$$

Some properties of the adjacency matrix are:

- it is a positive symmetrical matrix, i.e. $w_{i,j} = w_{j,i}$, and $w_{i,j} \geq 0$
- diagonal entries are zero, thus self-node similarities are avoided

We will use similarity and adjacency matrix interchangeably in this thesis and distances between the same nodes are set to zero.

3.3.2 Similarity graphs

Similarity functions are used to build different graphs. These graphs can be called similarity graphs and are derived from the adjacency matrix, $\mathbf{W}$. Although there might be other types of similarity graphs in different literature, we will discuss only ε-neighborhood, k-neighborhood, and fully-connected graphs [6].

ε-neighborhood graph

ε-neighborhood graph sets a threshold ε which determines the distances ranges that are selected or left out of the adjacency matrix [6].

The threshold ε can be chosen in order of $\left(\frac{\log(n)}{n}\right)^d$ for RBF kernel, where n is the number of samples, and d is the dimensionality of the data as this range provides the needed connectivity [6].

In anomaly detection, pairwise distances between the samples of the data are important in detecting outliers. Thus, placing a threshold on the distances might remove the samples that are dissimilar which might be anomalous.

k-neighborhood graph

The k-neighborhood graph, also called k-nn graph, constructs the similarity graph based on the k-nearest neighbors of each node [12].

This graph selects the number of nearest neighbors based on their distance, while ε-neighborhood graph sets only the distance of the neighbors but not the number of nodes, where the choice of k is in the order of $\log(n)$ [6].

The k-nn graph produces a sparser representation of similarities than the full adjacency matrix, $\mathbf{W}$ [6]. This representation might exclude samples which are not in the nearest neighborhood range, which makes these graphs

unpopular for anomaly detection task.

Fully connected graph

k-nn and ε-neighborhood graphs use thresholds to select distances used for defining similarities. Fully connected graphs [6] contain the entire adjacency matrix $\mathbf{W}$ without any thresholds.

In machine learning tasks, for instance in anomaly detection, it is usually important to examine similarities or differences that exist among the entire samples. For this purpose, fully connected graphs might provide better solution than the k or ε-neighborhood graphs.

In the previous sections, we have introduced background knowledge which help construct a graph and study its properties. In the next section, we study the different properties of graphs.

3.4 Graph properties

3.4.1 Node properties

Node degree

Node degree is a term describing how many samples are incident on or distributed around a particular node [13]. Node degree is the sum of the elements along the rows of the adjacency matrix, $\mathbf{W}$ [6]. Usually the node degree is represented by d_i, where

$$d_i = \sum_{j=1}^{N} w_{i,j}, \text{ where } \mathbf{W} = \{w_{i,j}\}, \text{ for } i, j = 1...n$$

If there are n nodes in a graph, $\mathbf{d}$ is $n \times 1$ a column matrix. $d_i \in \mathbf{d}$ shows the distribution of samples around node i.

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

Node degree can be used as one measure to analyze the connection among different nodes and the structure of the graph. Nodes with higher degrees will have more incident edges than nodes with lower node degrees [14]. The degree of nodes can be used to study similarity of the nodes by visualizing how samples are distributed around each node.

The definition of node degree is different for directed and undirected graphs. If the graph is undirected, there are no edges entering and exiting that node and the original definition of node degree will be used.

In directed graphs, there are incoming and outgoing edges to and from a node each having different edge values, $w_{i,j}$. Thus, the node degrees have to be defined separately as incoming and outgoing node degrees, and the the total node degree will be [14]

$$d_i = d_i^{in} + d_i^{out}$$

Node degree matrix

The node degree matrix, $\mathbf{D}$, is a diagonal square matrix whose diagonal entries are the node degree values d_i and the rest off-diagonal elements zero. The node degree matrix has the form:

$$\mathbf{D} = \begin{bmatrix} d_1 & & & \\ & d_2 & & 0 \\ & & \ddots & \\ & 0 & & d_{n-1} \\ & & & & d_n \end{bmatrix}$$

Node degree distribution

The term node degree distribution, P_d, defines the probability that a randomly chosen node has a node degree value d_i, which similarly can be expressed as the fraction of nodes in the network having the node degree value d_i. The incoming and outgoing node degree distributions of directed graphs can be separately calculated using their respective node degrees [14].

Size of sub-vertices

Size of sub-vertex $S_v \subset \mathbf{V}$ equals the sum of the degrees for all the vertices in S_v [6, 15]. It can be represented as:

$$Vol\,(S_v) = \sum_{j \in S_v} d_j$$

3.4.2 Disparity

Disparity is a term used for describing the dominance of a certain node over other sets of nodes [14]. This can be formulated from the adjacency matrix and the node degree as:

$$y_i = \sum_{j \in N_i}^{n} \left(\frac{w_{i,j}}{d_i}\right)^2,$$

where N_i defines the first *i-th* neighbors of the similarity matrix W. If fully-connected graphs are considered, the neighborhood extends over all the neighbors (the entire nodes).

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Each row i of $\mathbf{Y}$ describes the dominance of the node i over the rest of $n-1$ nodes.

Disparity might be useful in understanding the nature of individual nodes with respect to all other nodes or some group of nodes in the network.

3.4.3 Random walk matrix

Random walk matrix of a graph is a matrix whose entries $p_{i,j}$ define the probability of transition from node v_i to node v_j in one step [11, 16]. It is usually represented with matrix

$$\mathbf{P} = \begin{bmatrix} p_{1,1} & \cdots & & p_{1,n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ p_{n,1} & \cdots & & p_{n,n} \end{bmatrix}$$

In short, we can write $p_{i,j} = Prob\,(v_j = j \mid v_i = i) = \frac{w_{i,j}}{d_i}$, where $d_i = \sum_j w_{i,j}$ is the degree of node v_i.

The random walk matrix can be derived for weighted graphs using the relation [11]:

$$\mathbf{P} = \mathbf{D^{-1}W}$$

The matrix $\mathbf{P}$ has the following properties:

- $p_{i,i} = 0$ for graphs with no cycles, that is no transitions to same nodes
- the rows sum to one [15]
- its eigenvalues λ_i are such that $\lambda_1 = 1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq -1$ [15]

3.4.4 Graph Spectrum and Laplacian

The *graph Laplacian* $\mathbf{L}$, also called the *combinatorial Laplacian* [18], is the difference of the node degree matrix $\mathbf{D}$ and the adjacency matrix $\mathbf{W}$ [17]. Mathematically, it is expressed as:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

The *normalized Laplacian* matrix $\mathbf{L_n}$ is formulated as:

$$\mathbf{L_n} = \mathbf{D^{-1/2}LD^{-1/2}}$$

Both $\mathbf{L}$ and $\mathbf{L_n}$ are full $n \times n$ matrices.

The *graph spectrum* is defined as a matrix containing eigenvalues λ_i of the graph Laplacian, $\mathbf{L}$ [17].

Some important properties of the graph Laplacian and spectrum include [17]:

- $\sum_i \lambda_i \leqslant n$ iff $\mathbf{G}$ has no isolated vertices
- The spectrum of $\mathbf{G}$ is the union of the spectrum of the connected components in $\mathbf{G}$

Fiedler Vector

If one arranges the eigenvalues of the graph Laplacian $\mathbf{L}$ from the smallest to the largest, such that $0 \leqslant \lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_{max}$, the eigenvector associated with λ_2, the second smallest eigenvector, is called the *Fiedler vector* [19].

3.4.5 Centrality

Centrality is a measure describing importance or relevance of nodes in a network [20]. The network can be either a weighted or unweighted graph.

The concept of centrality was originally used in social graphs to analyze the interaction between agents (actors) in social networks but now has been used in different kinds of complex networks [21]. It can be one of the following: degree, betweenness, closeness and eigenvector centrality.

Degree centrality

Degree centrality of a vertex, abbreviated $C_D(v_i)$, measures the importance of a single vertex in the graph.

It can be calculated for a node v_i with degree d_i and the number of vertices, $|\mathbf{V}|$ [22] as:

$$C_D(v_i) = \frac{d_i}{|\mathbf{V}|-1}$$

A node will be called central if it has higher degree value. Nodes with relatively lower degree values have weaker link to the network than the central nodes [22].

Betweenness centrality

Betweenness of a node v_i, also called load [14], is the ratio of the shortest paths σ_{n_1,n_2} that pass through the node v_i to the shortest paths between nodes σ_{n_1,n_2} [14, 22, 23]. This can be written as [23]:

$$C_B(v_i) = \sum_{n_1 \neq n_2 \neq v_i} \frac{\sigma_{n_1,n_2}(v_i)}{\sigma_{n_1,n_2}}$$

The shortest path from n_1 to n_2 should have the node v_i neither as a starting nor an ending point. The path can be calculated using breadth-first search or other standard algorithms [14].

Closeness centrality

The *closeness centrality*, $C_c(v_i)$, is the reciprocal of the shortest distance between the vertices of a graph [23].

For node $v_i \in \mathbf{V}$, it is defined as [23, 24]

$$C_c(v_i) = \frac{1}{\sum_{n_t \in \mathbf{V}} d_G(v_i, n_t)}$$

where $d_G(v_i, n_t)$ is the shortest distance between the nodes v_i and n_t.

Eigenvector centrality

Eigenvector centrality uses the largest eigenvector, l_k, of the adjacency matrix $\mathbf{W}$ as a measure of relevance [25, 26]. It is computed from the adjacency matrix $\mathbf{W}$ and the largest eigenvalue λ_k of $\mathbf{W}$ as:

$$(l_k)_j = \frac{(\mathbf{W} l_k)_j}{\lambda_k}$$

Google [27] used a variant of eigenvector centrality to create an algorithm called `PageRank` that ranks web pages and hyper-links pointing to and from different web pages [28]. `PageRank` uses power iteration to solve the eigenvector centrality equation on a modified form of the adjacency matrix [26].

Chapter 4

Graph based clustering

4.1 Introduction to unsupervised learning

Unsupervised learning is a paradigm in machine learning which does not utilize any label information in the data.

The goals of unsupervised learning include discovering similar groups or clusters in the data (clustering), formulating the underlying distribution generating the data (density estimation), or visualizing the data onto a lower dimensional space [29].

The following sections focus on unsupervised clustering methods with more emphasis on graph based spectral clustering methods. In the last section, a new graph based clustering algorithm will be proposed.

4.2 Clustering methods

There are different types of clustering methods. Here we discuss mostly used clustering methods in machine learning: k-Means Clustering.

4.2.1 k-Means Clustering

The main goal of k-means clustering is partitioning a data set into groups of similar patterns. If we have a dataset of d-dimensional vector $\mathbf{X}$, where

$\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ where n is the number of samples, then k-means clustering partitions $\mathbf{X}$ into k groups.

Points inside a cluster have smaller distances than different points within different clusters, that is intra-cluster distances are smaller than inter-cluster distances. The number of clusters is usually unknown and has to be known in the beginning of the clustering [29].

The number of clusters, k, can be estimated using certain optimization criterion, or from some expert prior knowledge about the dataset. Then each $k_i \in |k|$ can be represented with a prototype vector μ_{k_i} where $k_i = 1, 2, ..., k$ and μ_{k_i} is d dimensional [29].

k -means clustering has an objective function $\mathbf{J}$:

$$\mathbf{J} = \sum_{N=1}^{n} \sum_{k_i=1}^{k} \left(r_{N,k_i} \left\| \mathbf{x}_N - \mu_{k_i} \right\|^2 \right)$$

where,

$$r_{N,k_i} = \begin{cases} 1 & \text{if } \mathbf{x}_N \in k_i \\ 0 & \text{otherwise} \end{cases}$$

which tries to minimize the euclidean distance of a sample $\mathbf{x}_N$ from a prototype vector μ_{k_i}. This will assign the sample to the k_i-th cluster. Thus, a sample point has to be in one of $k_i \in |k|$ clusters making the k-means a hard clustering algorithm.

4.2.2 Other Unsupervised Learning Methods

There exists other clustering methods like hierarchical clustering [30], co-clustering which clusters columns and rows of the data at the same time. This helps to know which rows and columns exhibit similar patterns. In addition, there are models described as local [31]. These models process the data in specified regions and find fits in that region. The input samples are localized in to groups (clusters). The models are collectively called competitive learning methods and include Self Organizing Maps (SOM), On-line k-means and Adaptive Resonance Theory (ART).

4.2.3 Graph based spectral clustering

Graphs have been introduced in Chapter 3 as tools of data analysis. In this section, we describe how to cluster data from a graph. In the following subsections we discuss some existing spectral clustering algorithms.

Graph based clustering in general refers to approaches that cluster the data based on the graph spectrum, which refers to the eigenvectors of the graph Laplacian.

Shi-Malik: Normalized Cut

This method was first introduced by Shi and Malik [32]. The idea of their method is to partition vertices $\mathbf{V}$ of a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ into sets $\mathbf{V_i}$ and $\mathbf{V_j}$ such that $\mathbf{V_1} \cup \mathbf{V_2} = \mathbf{V}$ and $\mathbf{V_1} \cap \mathbf{V_2} = \varnothing$.

They call the method of graph partitioning `Ncut` which stands for *normalized cut*. A cut of a graph is defined as the degree of dissimilarity between the partitions $\mathbf{V_1}$ and $\mathbf{V_2}$ measured with all the edges between the partitions removed [32]. Mathematically,

$$cut\,(\mathbf{V_1}, \mathbf{V_2}) = \sum_{v_i \in \mathbf{V_1}, v_j \in \mathbf{V_2}} w_{i,j}$$

The theoretical background and proofs are provided in the publication of Shi and Malik [32]. Algorithm 1 describes the *normalized cut* algorithm.

Algorithm 1 Normalized Cut for graph partitioning [32]

1. Construct a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ from the data, where the weights (edge values) measure the similarity of the nodes
2. Solve the eigenvectors of $(\mathbf{D} - \mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$
3. Use the eigenvector associated with the second smallest eigenvalue to bipartition the graph
4. If necessary, partition the segmented parts recursively

Step 3 of algorithm 1 partitions the graph into two subgraphs based on the sign of the second smallest eigenvector (Feidler vector) where positive values are in one cluster and negative values are in the other cluster, its median value or by searching for the best partitioning value with the best normalized cut [32].

There have been different variants of the normalized cut method suggested by different authors. Von [6] uses a k-means algorithm on the eigen decomposition of step 2, by selecting k columns of the eigenvectors and then performing the k-means clustering on the chosen eigenvectors.

Algorithm 2 Shi-Malik with k-means clustering [6]

1. Construct a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ from the data, where the weights (edge values) measure the similarity of the nodes
2. Solve the eigenvectors of $(\mathbf{D} - \mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$
3. From the generalized eigen-decomposition of step 2, choose the first k columns of the eigenvectors from the decomposition in step 2
4. Perform k-means clustering on each row of the chosen eigenvectors

There have been different approaches to spectral clustering method.
In [33], a new algorithm for spectral clustering with an objective function that minimize the error measure between a given partition and the minimum normalized cut partition is suggested.
In [34], a similar approach with algorithm 2 is used except the authors normalize the rows of the chosen eigenvectors of step 3 to norm 1.
In [35], the authors present kernel k-means algorithm and derive the *nominalized cut* algorithm as a special case of the kernel k-means.
In [36], the authors suggest a new K-way clustering method that is able to partition the data into K clusters.

Most of the mentioned spectral clustering algorithms are highly dependent on the eigenvector decomposition of the graph Laplacian and are variants of the Shi and Malik's method.

In the following section, we propose a new graph partitioning algorithm based on the node degree values without using eigenvalue decomposition on the Laplacian.

4.3 Proposed method

In Section 3.4, different properties of a graph are discussed. Centrality was one concept used to study properties of graphs or networks as mentioned in subsection 3.4.5. We use the degree centrality property of a graph to bipar-

tition the nodes into two disjoint sets of nodes. The choice of this measure is due to its easiness of computation.

We call our algorithm `NodeClustering`, since it uses the node properties to do the clustering. In the following subsections, we explain `NodeClustering` method and show the method with an example.

4.3.1 `NodeClustering`: Node Bi-partitioning Algorithm

This section explains the main work of the thesis. Algorithm 3 list the steps of the method.

The first step is constructing a weighted graph from the given data $\mathbf{X_N^d}$, where N represents the observations and d represent the dimensions. In the data, there are different feature values and there are variations in the individual values. If one wants to compare these values, features have to be on similar footing. Scaling and normalization are preprocessing method to remove such effects and make features comparable. Logarithmic scaling is one example [37].
The pairwise distances between the scaled samples can be computed using the euclidean, Jaccard or other methods. Applying one of the kernels on the pairwise distances in table 3.1 will give the similarity matrix of the samples,$\mathbf{W}$.

If $\mathbf{x_i}$ and $\mathbf{x_j}$ are samples from the data $\mathbf{X}$, the pairwise similarity between the samples using the RBF kernel can be calculated as:

$$w_{i,j} = \exp(-\sigma \times \|\mathbf{x_i} - \mathbf{x_j}\|^2),$$

where σ is the scaling parameter for the kernel and its inverse $\frac{1}{\sigma}$ is the width of the kernel [29].

The similarity matrix $\mathbf{W}$, will be $n \times n$ matrix containing the pairwise similarity of all the samples, and the diagonal of this matrix is set to zero, as the distance between the same features is zero. The node degrees were computed by summing the rows of $\mathbf{W}$ as shown in step 3. This gives the individual node degree values in a column matrix $\mathbf{d}$,

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

Algorithm 3 Node-Clustering Algorithm: `NodeClustering`

Input:Data $\mathbf{X}$, threshold of partition $\boldsymbol{\tau}$

Output: Set of nodes $\mathbf{V_1}$ and $\mathbf{V_2}$ such that $\mathbf{V_1} \cup \mathbf{V_2} = \mathbf{V}$ and $\mathbf{V_1} \cap \mathbf{V_2} = \varnothing$

1. Normalize the data to $\mathbf{X_N}$, for example using logarithmic scaling
2. Construct the pairwise distance between the samples of $\mathbf{X}$, and form the adjacency matrix $\mathbf{W}$ {this creates a weighted undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where nodes are in $\mathbf{V}$ and edge values are in $\mathbf{W}$}
3. Calculate the node degrees $d_i = \sum_j w_{i,j}$, where $\mathbf{W} = \{w_{i,j}\}$; $\mathbf{d} = \{d_i\}$, where $i = \{1, \ldots, n\}$.
4. Initialize $\boldsymbol{\tau}$ with some value in the value range of $\mathbf{d}$
5. **Output** $\mathbf{V_i} = \{i : d_i < \boldsymbol{\tau}\}$, $\mathbf{V_j} = \{j : d_j \geq \boldsymbol{\tau}\}$

Each $d_i \in \mathbf{d}$ can be interpreted to signify the similarity around the node v_i with respect to other nodes in the graph.

If some set of nodes $\mathbf{V_i} \in \mathbf{V}$ have their $d_i \in \mathbf{d}$ values closer than other set of nodes $\mathbf{V_j} \in \mathbf{V}$, then original features in the node set $\mathbf{V_i}$ will have more closer pairwise similarity values in the similarity matrix and hence in the original input space and are expected to have similar properties than the nodes in set $\mathbf{V_j}$.

A difficult question is setting the threshold $\boldsymbol{\tau}$ which sets the limits of partitions on the individual node degree values.

4.3.2 Why `NodeClustering` work?

In this section, `NodeClustering` is demonstrated with an example toy data.

One reason why `NodeClustering` is valid is the use of the kernel transformation from the input space $\mathbb{R}^{\mathbf{d}}$ to $n \times n$ kernel matrix. This transformation will

make the constructed adjacency matrix capture similarities more effectively than in the original space.

The second argument is the nature of the similarity matrix. The entries of the matrix contain pairwise similarity between the individual feature values. The similarity matrix also provides the node degrees, which are the sum of the rows of the similarity matrix.

The following subsection provides an example of a known background process generating the data and how well the `NodeClustering` algorithm discovers the generating process.

4.3.3 Example with a synthetic data

A simple demonstration of the node degree and how it captures the similarity of the samples is shown next for a synthetic data.

The data is generated from two different probability distributions, where the first ten samples of the data are from a random normal distribution with zero mean and 0.5 variance. The rest ten samples are also from a random normal distribution with 0 mean and 0.25 variance as follows:

$$\mathbf{X_1} \sim \mathbf{randomNormal}\,(10, 3)\ ,\ \mathbf{X_2} \sim \mathbf{randomNormal}\,(10, 3)$$

A single dataset $\mathbf{X}$ comprising both sets is formed and let $\mathbf{X} = (\mathbf{X_1} \cup \mathbf{X_2})$. A weighted undirected graph is constructed from $\mathbf{X}$ using RBF kernel, using the scaling of the kernel as 0.5 , by concatenating $\mathbf{X_1}$ and $\mathbf{X_2}$ one after the other. The degrees of the 20 nodes are obtained by summing the rows of the adjacency matrix $\mathbf{W}$ of the graph. Figure 4.1 plots each of the node degree values with respect to the sample (node) it represents.

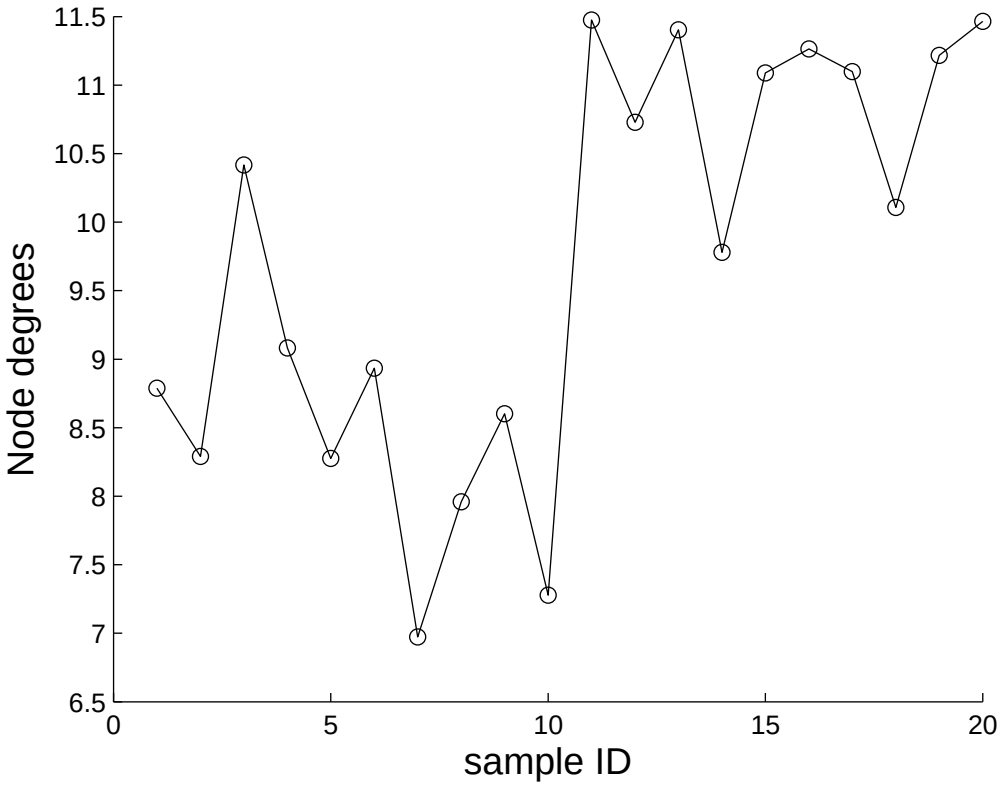

Figure 4.1: Node degree values of a synthetic dataset

The figure shows the first 10 nodes from the left (group 1) are in proximity to each other than the right most 10 nodes (group 2). There is a big jump (transition) in the degree values from node 10 of group 1 to node 11 of group 2. The center of the jump can be considered as a threshold that splits the graph into group 1 and group 2. Choosing the center value as the mean of the node degrees is one example of threshold. This threshold can identify samples generated by the same distribution.

The next chapter discusses application of `NodeClustering` for anomaly detection with two real world datasets.

Chapter 5

Anomaly Detection with Graphs

This chapter introduces two real network traffic datasets from IP networks, studies properties of graphs constructed from these data and applies the method `NodeClustering` for anomaly detection in these graphs. Finally, a thorough discussion of the results will be given.

5.1 Introduction to the data sets

In the thesis two traffic datasets were considered: NSL-KDD 99 dataset [44] and a set generated in Nokia Siemens Networks (NSN) laboratory, called NG-set. A brief description of the datasets with used preprocessing method and feature lists is given in the following subsections.

5.1.1 NSL-KDD 99 dataset

KDD-99 is one of the widely used and studied datasets in anomaly and intrusion detection research. This data has been used in validating and testing machine learning and data mining methods for various tasks, for instance in anomaly detection.

The original data, initially from a project of Defense Advanced Research project Agency (DARPA) [39], was collected for about two weeks simulating

Local Area Network (LAN) of a United States Air force military base [38]. Lincoln Laboratory [40] of the Massachusetts Institute of Technology (MIT) made the simulations in 1998. The NSL-KDD 99 dataset is a derivative of the DARPA 98 dataset.

The original task in the KDD-99 cup competition was to build a supervised learning method that can classify samples of the data, also called connections, into attack and normal labeled samples. Thus, the data has training and testing sets available for learning a model and evaluating the designed method. The training set has labels describing each observation as normal or attack. However, the data has also been used in developing unsupervised learning methods which do not use the label information in any manner.

Despite the popularity of the NSL-KDD 99 dataset in the machine learning community, there have been publications describing its shortcomings. Problems in the set-up that generated the data among other problems are listed in [43]. Moreover, problems of redundant samples in the training and testing set are mentioned [44] and the authors provide a new data derived from the KDD-99 by removing the repetitions, called NSL-KDD [45]. NSL-KDD data is considered a better option than the original KDD-99 set as it solves some of the problems of the original NSL-KDD 99 data.

In this thesis, the NSL-KDD 99 set was used. In addition to the aforementioned problems, this set still faces problem of outdatedness.

Preprocessing

In network traffic data analysis, it is possible to use either packets or flow data for different analysis and detection tasks.

The difference between flows and packets is that a packet contain all the information sent from the source to the destination, where as a flow summarizes the packets with in the same connection. These details in the flows are considered as features for the traffic data. There are also different tools that can convert between these data formats.

Argus tool [41] is one example which can convert packets into flows. No conversion is required as the flows were already given in [45]. A smaller sample

was randomly chosen from the training data to make the experimental data for this thesis. It should be noted here that it is equally possible to use the testing data as the source of the experimental data.

The experimental data contained 10 % attack and the remaining 90 % of attack-free (normal) sample. The structure of the data is that all samples from normal traffic are followed by the attack traffic. This structure is chosen to observe any immediate changes of pattern in the graph.

Features

The NSL-KDD 99 dataset has a complete description of features listed in [38]. There are a total of 41 features, of which 7 are discrete and the rest 34 are continuous-valued. There was one column of *the number of outbound commands* feature that was entirely zero and was removed. This reduced the total continuous features to 33 listed in Appendix A.1.

5.1.2 NG-set

The original KDD data was from 1999, or exactly in 1998 when the DARPA dataset was collected, and certainly lacks the capability of representing the current threats and attacks in the world of network security. There are newer services, protocols, and applications that did not exist in 1999, with different patterns and behavior to the network traffic data. Taking these factors in to consideration, a new dataset from a small network setup in NSN was generated. This will be called the NG-set. The name initials signify the data is generated (G) from a network (N) not from an artificial simulator.

NG-set was collected from a small network of four hosts. There is one machine generating the attacks and other three machines doing normal activities, like browsing, sending email, making Facebook updates, phone calls on the Internet etc. The data set contained 10 Gigabytes of traffic data in 12 different capture files in packet capture (pcap) format. Due to the size of the data, only the last two capture files with a size of 730 megabytes were used in the thesis.

Metasploit [46] framework generated the attacks along with other smaller scripts that automated the attacks. The attacks mainly included port scanning, password guessing, vulnerability scanning and exploiting based on the

scans. In vulnerability scanning, different components of the systems for example software versions or system settings were checked and if associated vulnerabilities were found with the versions or the system settings, the system was attacked. In addition, password guessing checked if passwords for some service or application can be guessed easily. For instance, logging as root user using the same user name and password root is an example of a password guessing attack. Port scanning refers to sending crafted packets to specific ports in the scanned computer, and waiting for a reply. The goal is to learn whether there is a service running in a port (or services in a set of ports).

Preprocessing

The packet data was captured in a tool called Wireshark. The packet form of the data was unusable for the designed graph based clustering method since the packets do not have the features explicitly listed. Thus, the packet data was converted into flows.

Each flow represented a connection made from a source IP address to a destination IP address. The traffic data in captured pcap files contained information about the packets. The graph based method will require features from the flows. For this reason, a tool called *argus* [41] was used to convert the packets into flows. It has command line tools called `argus` and read argus (`ra`) that can filter the pcap files into flow transactions based on the packets original capture time [42]. Processing the packets into flows was done in Linux environment as it was easier to setup Argus in such environment. The argus tool first converts the captured traffic data in the pcap to a format readable by the read argus tool. This tool was used with some command line options which produced the NG-set. The data contained features that are interesting to the security analysis. The actual preprocessing commands are listed in Appendix C.

Features

The NG-set included discrete and continuous valued flow features. A total of 23 features were extracted from the flow data. Although Argus has more choice of features than those specified in Appendix B, some had a non-numerical value which led to their exclusion from further analysis. A complete list of the features is listed in Appendix B.

5.2 Traffic data analysis with Graphs

5.2.1 NSL-KDD 99 Dataset

Graph construction

NSL-KDD 99 data has huge number of samples. Graph construction with the original data will be computational expensive. This limitation was solved by randomly selecting a small traffic set with 1500 samples and 33 features.

The selected samples comprise majority of the traffic with no attacks (normal samples) and the rest with attacks. This resembles a normal operating network. It has to be noted that attacks are anomalies and furthermore the term anomaly can describe other events that diverge from normal pattern. Figure 5.1 shows the selected samples on the X-axis and their feature values on the Y-axis.

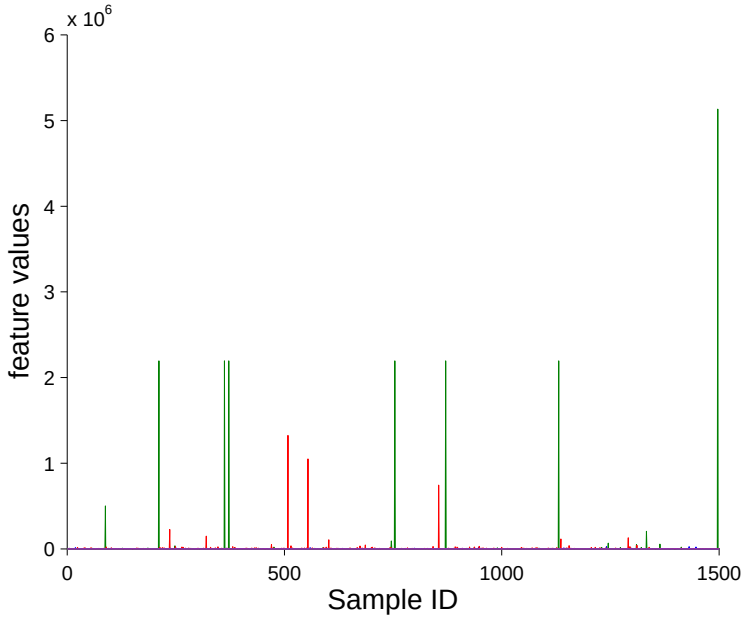

Figure 5.1: NSL-kDD 99 dataset

In figure 5.1, majority of the features have small values close to the X-axis, while few have higher values with large values.

The selected smaller set $\mathbf{X}$ contains 90% normal and the rest 10% attack traffic. Without loss of generalization, the attack samples are concatenated

after the normal samples and logarithmic scaling produces $\mathbf{X_N}$, where

$$\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_{1500}}\}^T, \mathbf{X_N} = \log(\mathbf{X} + 1),$$

The addition of the constant 1 avoids the problem of infinite values with feature values of zero.

RBF kernel was applied on the scaled data to construct the similarity matrix of the graph $\mathbf{W}$ which is 1500×1500 symmetrical matrix, where $\mathbf{W} = \{w_{i,j}\}$ and

$$w_{i,j} = \exp\left(-1 \times \delta \times (\mathbf{x_i} - \mathbf{x_j})^2\right)$$

expresses similarity among the samples $\mathbf{x_i}$ and $\mathbf{x_j}$.

Choosing the scaling parameter of RBF

RBF kernel has hyper-parameter δ that determines how pairwise similarities are scaled. The choice of this parameter has direct influence on the node degree values and volume of the graph as discussed in section 3.4.

Figure 5.2 compares the scaling parameter δ with the volume of the graph.

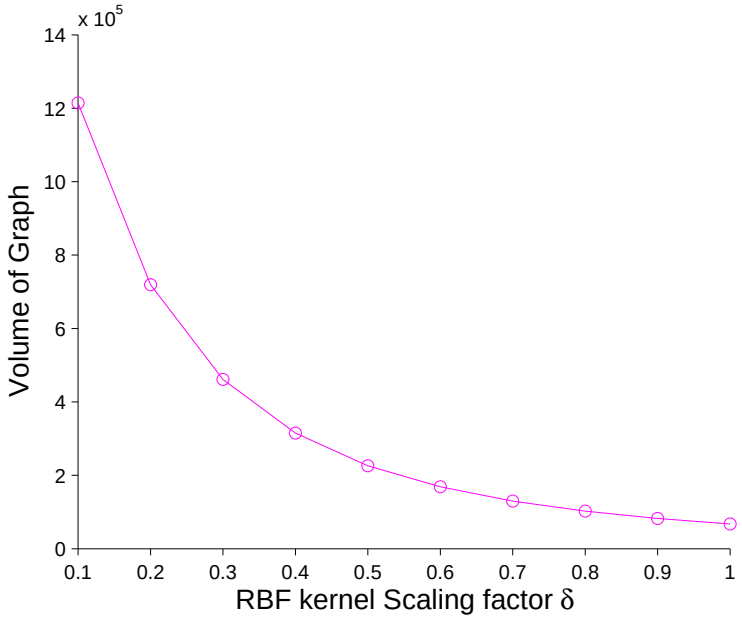

Figure 5.2: The effect of hyper-parameter δ on the size of the vertex

The scaling parameter δ was selected so that $\delta \in \{0.1, 0.2, ..., 1\}$. This choice was arbitrary and one could have chosen different values for this. δ has direct influence on the similarity matrix and on the node degree values. Volume of graph $\mathbf{G}$ with individual nodes of degree d_i can be calculated as:

$$Vol\,(\mathbf{G}) = \sum_{i \in \mathbf{G}} d_i \text{ [6]}$$

Figure 5.2 shows that the volume of the graph decreases with the increasing value of the scaling parameter, δ. The volume of the graph can describe how compact or dispersed the graph is.

If $\delta = 1$, then the RBF kernel is exactly the same as an exponential kernel and the resulting graph is dense. When δ decreases to 0.1, the graph's volume increases and graph becomes sparse.

The scaling parameter was chosen to be 0.5 with an assumption that such a graph will exhibit average properties of the dense and sparse graphs.

Properties of NSL-KDD 99 graph

Node degrees

The node degree d_i of node i is calculated so that

$$d_i = \sum_{j=1}^{j=1500} w_{i,j}$$

and for the entire nodes

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

The node degrees show the link that exists within the nodes of the graph. Nodes with closer feature values will have stronger link and the rest of the nodes will have weaker links to them. Figure 5.3 plots the individual node degrees (d_i) versus the sample node i represent.

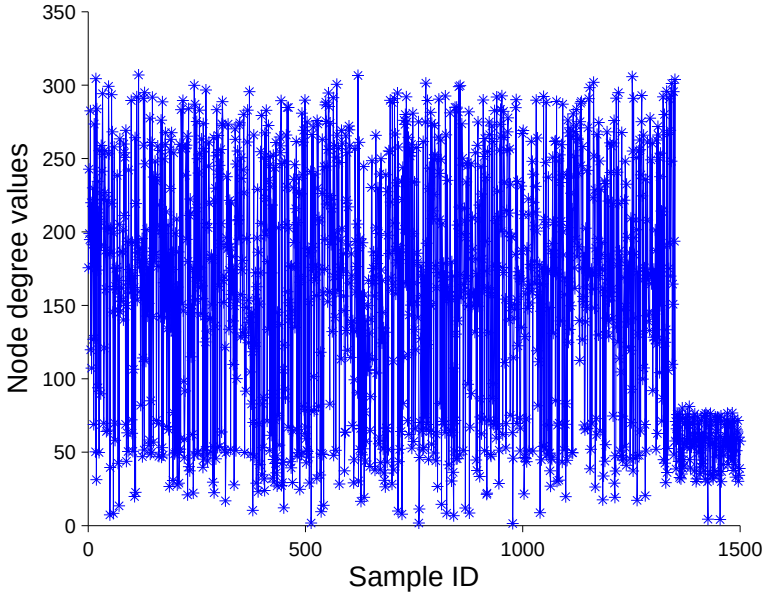

Figure 5.3: Node degree distribution in the NSL-KDD 99 dataset

The figure 5.3 shows nodes in the right most corner of the plot make a close and compact group. This group has lower degree node than the remaining nodes on the left. Moreover, there is a jump from the left group to the compact smaller node degree valued nodes showing that the nodes in the left most region have similar patterns than the other nodes. The node degree distribution provides more detail than provided by the original data set shown in figure 5.1.

The degree centrality has been presented in section 3.4. It is solely based on the node degrees and will signify the same property as in figure 5.3. From this figure, one can anticipate nodes in the right most group to behave differently than those in the leftmost group. This will be studied by considering few nodes in both groups using the random walk matrix.

Random walk matrices

The previous analysis with node degrees showed that there are visibly two groups in the data. The random walk matrix will give more information about the nodes with probability values.

Random walk matrix $\mathbf{P}$ has been discussed in section 3.4. One entry $p_{i,j} \in \mathbf{P}$ describes the probability of transitions from node i to node j. One can calculate $\mathbf{P}$, which is 1500×1500 symmetric matrix as:

$$\mathbf{P} = \mathbf{D}^{-1} \times \mathbf{W}$$

A new notation $\mathbf{P}(\mathbf{i}, \cdot)$ can be used to describe i^{th} row of $\mathbf{P}$. Adhering to the same interpretation as $p_{i,j}$, $\mathbf{P}(\mathbf{i}, \cdot)$ will represent the probability of transition from node i to the rest of the nodes in the graph. It should be noted that $p_{i,i} = 0$.

The analysis with the node degrees in figure 5.3 has shown there are two regions: a small compact region on the right (region 1) and the rest of the region on the left (region2). We pick two nodes from these regions and visualize how the transition probabilities change in these regions.

We pick node 1 from region 2 and node 1500 from region 1. Consequently, $\mathbf{P}(\mathbf{1}, \cdot)$ and $\mathbf{P}(\mathbf{1500}, \cdot)$ from $\mathbf{P}$ were plotted to visualize any changes in the

transition probabilities from node 1500 and 1 to the different regions. Figures 5.4 and 5.5 plot the transition probabilities from nodes of the different regions.

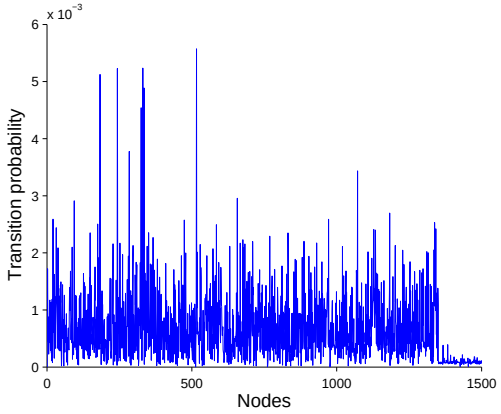

Figure 5.4: The first row $\mathbf{P}$ for NSL-KDD data set

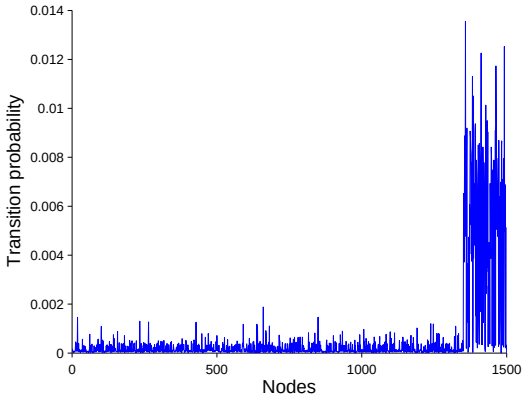

Figure 5.5: The last row of $\mathbf{P}$ for NSL-KDD data set

Figure 5.4 plots $\mathbf{P}(\mathbf{1}, \cdot)$, which is the transition probabilities from node 1 to rest of the nodes in the graph. The transition probabilities have higher values for the nodes in the left region (region 2) than for those nodes in right most region (region 1). This can be interpreted as the nodes in region 1 are more likely to be observed together and behave similarly than nodes in region 2.

Figure 5.5 plots $\mathbf{P}(\mathbf{1500}, \cdot)$, which is the transition probabilities from node 1500 to rest of the nodes in the graph. The transition probabilities are higher nodes in the right most region (region 1) than nodes in the left region (region 1). This can be interpreted as the nodes in region 1 are more likely to be observed together than nodes in region 2.

The analysis with the random walk matrix has shown that the nodes in the different regions have different transition probabilities, which signifies that the nodes have different behaviors when there are transitions from high to low or low to high probability values.

In addition to the above analysis, one can use other properties of the graphs to study the similarity among the nodes in the graph.

5.2.2 NG-set

The NG-set has two preprocessed feature sets as shown in Appendix C. From these sets, only set 1 (`NgSet1`) is used for simplicity. This set has about 4000 samples of which 2000 will be randomly chosen to construct a graph and study its properties.

With similar justifications to the NSL-KDD 99 dataset, the randomly chosen set contained 90% normal samples and the rest 10% attack samples. Moreover, only the continuous valued features are used for the analysis. Thus, fully connected, weighted and undirected graphs were constructed.

Graph construction

The samples in the data were logarithmically normalized and pairwise distances with RBF kernel gave the adjacency matrix. The scaling parameter δ was set with value of 0.5 with a similar argument given for the the NSL-KDD 99 dataset. Figure 5.6 shows the how the volume of the graph changes with the scaling parameter.

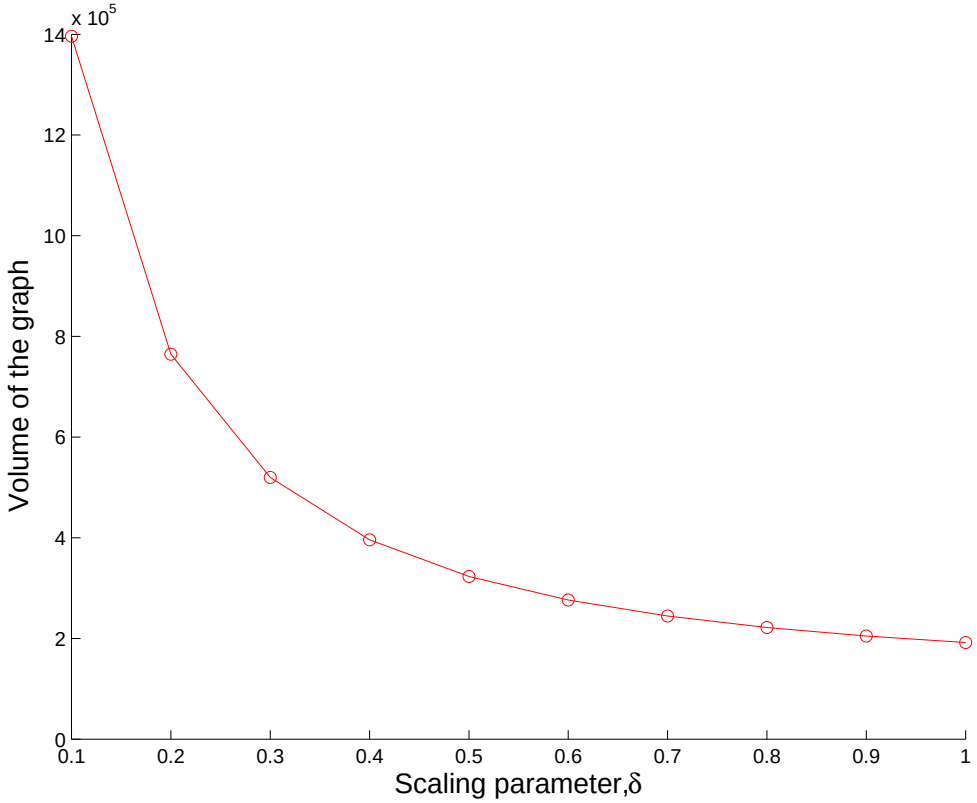

Figure 5.6: The effect of scaling on the volume of the graph

The volume of the graph exponentially decreased with increasing value of the scaling parameter. In the following subsections, node degree values and random walk matrix will be used to study the property of the NG-set graph.

Node degree

The node degrees were calculated by summing the rows of the adjacency matrix of the graph. Figure 5.7 shows how the nodes are distributed in the graph.

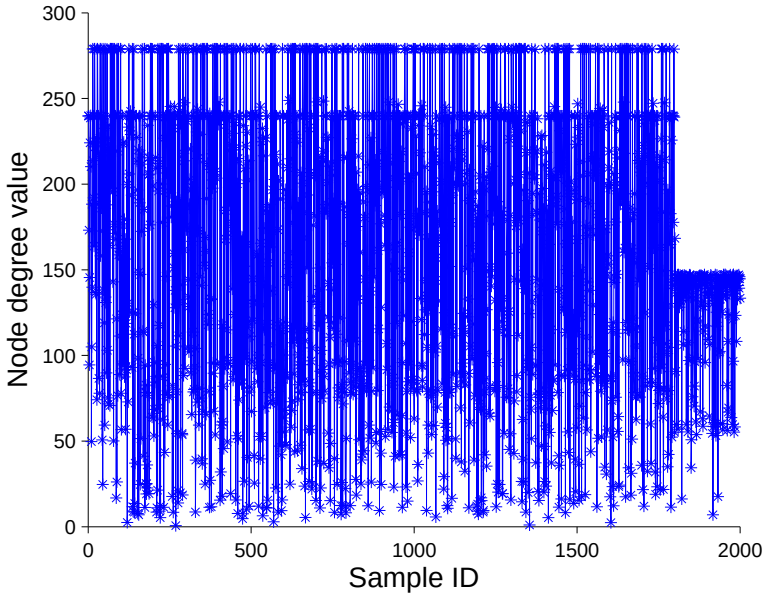

Figure 5.7: The node degrees of the NG-set

There is a variation of the values of the nodes. There are two regions in figure 5.7. The first region has generally higher node degree values (on the left most region)and the other region has lower node degree values which shows smaller values in some parts than other portions of the graph. This behavior similarly had been observed in the NSL-KDD 99 dataset. The change in the node degree values is further investigated with the transition probabilities in the random walk matrix.

Random walk matrix

Entries of the random walk matrix represent the transition probabilities from one node to another node. There are two regions observed in the node degree values in figure 5.7. Two nodes are picked from the left and right most regions of this figure and transition probabilities are visualized.

The entries of the rows of the transition matrix can be used to infer the probability of transition from the current node (which is the current row number) to another set of nodes excluding the current node. Figure 5.8 and 5.9 plot the first and the last row of the random walk matrix, $\mathbf{P}$.

Figure 5.8 shows the plot for the first row of the transition matrix, $\mathbf{P}(\mathbf{1}, \cdot)$.

The values show the transition probabilities from node 1 to the rest of the nodes, $2, 3, ..., 2000$. In this figure, the transition probabilities have higher values in most of the left region and lowest values in the right most regions. This shows that nodes in the left regions have higher probabilities of being observed together with node 1 than the nodes on the left region.

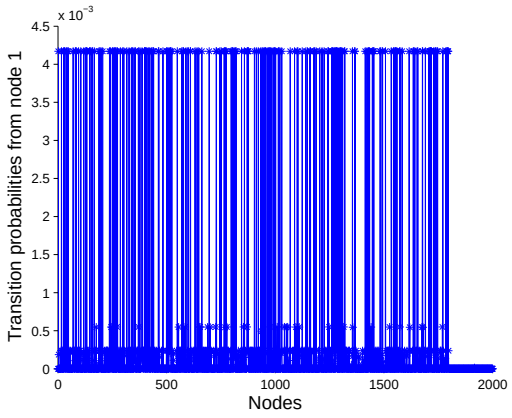

Figure 5.8: The first row of the transition matrix P of NG-set

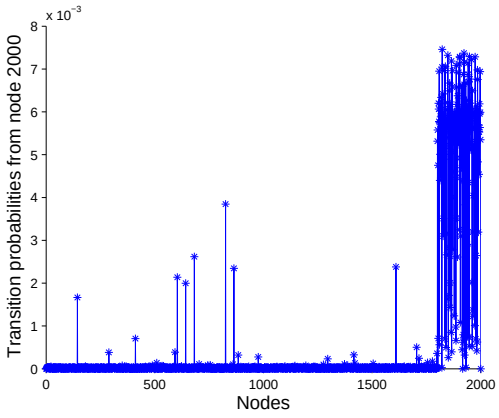

Figure 5.9: The last row of P for NG-set

Figure 5.9 visualizes the last row of the transition matrix, $\mathbf{P}(\mathbf{2000}, \cdot)$. The entries of this row represent the transition probabilities from node 2000 to nodes $1, \ldots, 1999$. This plot shows that there are stronger transition probabilities from node 2000 to the nodes in right most region than most of the nodes in the left region. In the left region, more nodes have lower transition probabilities of being observed with the node 2000. In addition, there are few nodes in the left region where there are higher probabilities of transitions to node 2000.

The analysis with undirected weighted graphs on the network traffic datasets have shown that there are regions of different node degrees and transition probabilities. Both of these analysis have discovered similar regions of nodes that showed similar variations.

In the next section, `NodeClustering` is applied to both datasets for partitioning the graphs into more general regions, which include the nodes that might have similar node degree values or transition probabilities but are in opposite regions.

5.3 Anomaly detection with `NodeClustering`

In Chapter 4, `NodeClustering` was introduced for graph bi-partitioning task. In this section, a slightly modified version of this algorithm (listed in algorithm 3) is used to perform graph clustering for anomaly detection task.

Algorithm 4 describes how the proposed node clustering method can be used in anomaly detection task for network traffic data. The rest of this section describes design decisions made in the different steps of the algorithm.

Network data can have continuous and discrete valued features. Thus, a graph constructed from the features can be either weighted or unweighted. The graphs from the continuous and discrete features are called continuous and discrete graphs respectively. In this thesis, weighted continuous graphs have been used, because majority of the features in both datasets are continuous valued as shown in Appendix B and [38]. Hereafter, the weighted continuous valued graphs will be named as graphs for simplicity

Algorithm 4 Node-Clustering Algorithm: `NodeClustering`

Input: Traffic data $\mathbf{X}$, threshold of partition $\boldsymbol{\tau}$
1.Select the continuous features from the data, $\mathbf{X_C}$
2. Normalize $\mathbf{X_C}$
3. Construct the pairwise distance in $\mathbf{X_C}$, and form the adjacency matrix $\mathbf{W}$ {this creates the weighted undirected graph}
4. Calculate the node degree $d_i = \sum_{\forall j} w_{i,j}$, where $\mathbf{W} = \{w_{i,j}\}$, and $\mathbf{d} = \{d_i\}$, $i = 1 \dots n$
5. Initialize $\boldsymbol{\tau}$ with the mean of the node degree, d_μ
6. **Output** $\mathbf{A} = \{i : d_i < \boldsymbol{\tau}\}$, $\mathbf{N} = \{i : d_i \geq \boldsymbol{\tau}\}$

The other steps of the algorithm are basically the same as described in algorithm 3. On the other hand, the threshold of separation has been set to the mean of the node degree values in d_i. This choice is a heuristic based.

The threshold $\boldsymbol{\tau}$ separates normal nodes from anomalous ones and assign the nodes into two such disjoint sets. An important question is which nodes are assigned to which sets. Since we do anomaly detection in a normally operating network traffic, data from such network contains higher number of normal and fewer anomalous samples. In such networks, it is expected that

many of the samples are highly similar and few deviate from this similarity.

The similarity matrix, $\mathbf{W}$, is computed with RBF kernel. It will have higher value for $w_{i,j}$ if samples i and j have higher pairwise similarity. In the normally operating network, majority of the samples will have high similarity values and the remaining anomalous samples will behave differently from the normal ones and will be less similar to the normal nodes. Thus, nodes with equal or higher degree nodes than $\boldsymbol{\tau}$ will be assigned to the normal set $\mathbf{N}$ and the remaining nodes will be assigned to the anomalous set $\mathbf{A}$.

The evaluation of `NodeClustering` was made using the indicators in table 5.1, which shows the matrix used for evaluating the performance. The specificity and sensitivity values will be particularly used to evaluate the algorithm. True labels of the NSL-KDD 99 and the NG-set have been provided. These have been used in table 5.1 for evaluation purposes.

	Real Anomalies	**Normal Traffic**
Reported Anomaly Set	True Positive (TP)	False Positive(FP)
Reported Normal Set	False Negative(FN)	True Negative (TN)

Table 5.1: Evaluation matrix

From the different values in table 5.1, the true positive (sensitivity) and false positive rates $(1 - specificity)$ will be derived as

$$True\ Positive\ Rate = sensitivity = \frac{TP}{TP+FN}, \text{ and}$$
$$False\ Positive\ Rate = 1 - specificity = 1 - \left(\frac{TN}{TN+FP}\right)$$

Sensitivity defines the percentage of anomalies that are correctly identified as anomalies by the algorithm from the total known anomalies in the data, while specificity defines the percentage of normal traffic that is correctly identified as normal from the total known normal traffic data.

The datasets were chosen so that we randomly split set of 1500 samples from the original NSL-KDD 99 set and 2000 samples for the NG-set. The random splitting was done 20 times to make better performance measure and average values of the different splits were taken. Graphs were constructed for each split and for every run we measure the performance values. For all the splits, the threshold $\boldsymbol{\tau}$ was set to be the mean of the node degree values $\mathbf{d}$.

Table 5.2 shows the sensitivity and specificity values obtained from algorithm 4.

Data	Sensitivity	Specificity
NSL-KDD 99	0.9990	0.6702
NG-set	1.000	0.5516

Table 5.2: Performance of `NodeClustering` on NG-set and NSL-KDD 99

The results in table 5.2 show that `NodeClustering` has sensitivity of almost 100% for both datasets. This means that it successfully identifies all the known intrusions in the data.The specificity value shows that `NodeClustering` will identify 67.02% of the normal traffic as normal and mis-classify 32.98% of the normal samples as intrusive. In the case of the NG-set, `NodeClustering` will recognize 55.16% of the normal traffic as normal and the remaining 44.84% as intrusive although it is labeled normal in the data.

Grouping normal traffic to the intrusive set might be less dangerous than assigning intrusive sample to the normal set. Moreover, the labels in the given dataset are of only attack labels and `NodeClustering` finds out anomalous nodes which include attacks samples, samples from mis-configured network components or any samples that deviate from the normal traffic data pattern. This section showed `NodeClustering` successfully identified the known intrusions to the anomalous set in the studied datasets, and achieved graph bi-partitioning to accomplish the anomaly detection task.

In the next section, we discuss the validity of `NodeClustering` results and compare its performance with the state of the art spectral clustering method.

5.4 Discussion of the results

In the previous section, the results of the `NodeClustering` algorithm had been shown. We used RBF kernel with scaling parameter of 0.5 and showed how the volume of the graph changed with this parameter. In addition, we set the threshold of separation between normal and anomalous samples as the mean of the node degrees.

In the coming subsections, we will evaluate how the performance of the `NodeClustering` algorithm varies with the scaling parameter of the kernel δ, the threshold of separation $\boldsymbol{\tau}$. In addition, we make comparisons with two variants of a spectral clustering method.

5.4.1 Effect of scaling parameter δ

`NodeClustering` constructed weighted undirected graphs using the RBF kernel

$$w_{i,j} = exp(-1 \times \delta \times \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

where δ is the scaling parameter. This parameter has the effect of scaling the pairwise distances between the samples and affects also the volume of the graph. We show how changing its value affects the performance of `NodeClustering` method.

Figure 5.10 plots the performance measures versus the width hyper-parameter for the NSL-KDD 99 dataset. The scaling parameter has been chosen in the range $0.1, \ldots, 1$ for this demonstration and for every threshold value the average of 30 runs for the estimation of the performance values has been done.

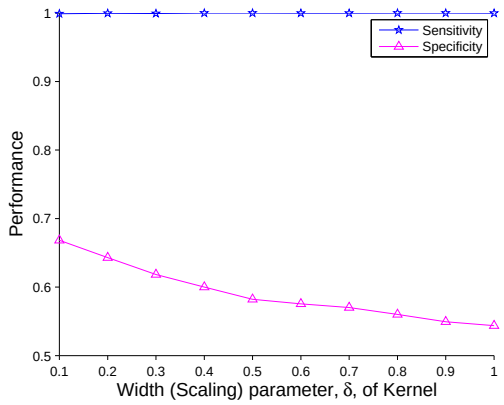

Figure 5.10: Effect of scaling parameter δ on the FPR and TPR of NSL-KDD 99 dataset

This figure shows increasing the scaling parameter does not affect the sensitivity at all, while the specificity drops by almost 10% on increasing the scaling, which means `NodeClustering` will recognize fewer normal samples

with higher scaling parameter than with the smaller scaling parameter. Thus, increasing the width results in larger false positive rate while smaller values of δ will result in a smaller false positive rate. Moreover, decreasing or increasing the scaling has no significant effect on the true positive rate. In KDD-NSL 99 set, this analysis provides an evidence that compact smaller volumed graphs discover intrusions with the same true positive rate as the larger volumed graphs, but with a smaller positive rate. We apply the same procedure as above for the NG-set. Figure 5.11 shows the results.

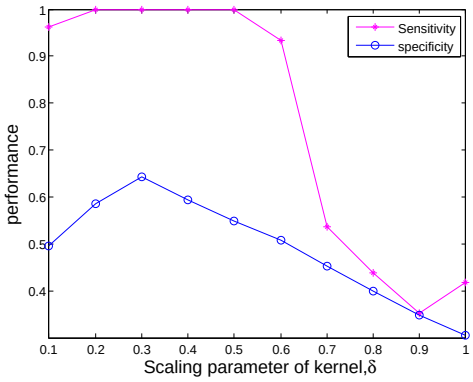

Figure 5.11: The effect of the width hyper-parameter on the TPR and Specificity of NG-Set

This figure shows the sensitivity of `NodeClustering` increase rapidly and then start decreasing with increasing of the scaling. In addition, the specificity also follows similar trend. The increases in the sensitivity and specificity are observed when the scaling parameter is smaller. This means that the graph is denser at this scale than at higher scales.

The analysis with the scaling parameter of the kernel showed that it is good to use `NodeClustering` algorithm with smaller scaling values to get better performance. This might be due to the fact that volume of the graph is really small and anomalies can easily stand out from the dense volume of the graph easily.

5.4.2 Effect of threshold τ

`NodeClustering` uses the threshold τ for bi-partitioning graph nodes either as normal or anomalous. We used heuristics and set its value to the mean of the node degree values, but now we use different values for this threshold and study its effect on the performance of `NodeClustering`.

We chose different scales for the thresholds. The factoring scale was varied from $0.1 \times d_\mu$ to d_μ. This scaling decreased the threshold values by 90% maximum. Figure 5.12 plots the effect of decreasing the threshold below the mean of the node degree on the NSL-KDD 99 set.

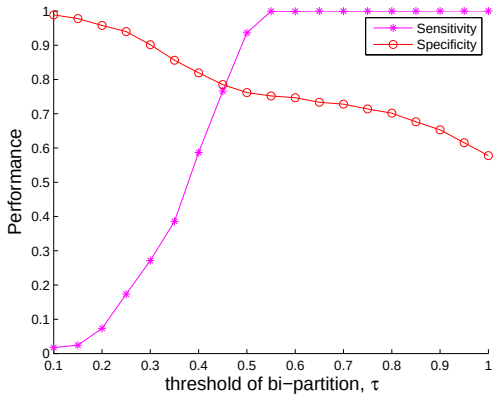

Figure 5.12: Effect of threshold τ on performance of NSL-KDD 99 set

This figure shows that increasing the threshold τ keeps the sensitivity increas-

ing and remains stable after $0.6 \times d_\mu$ till the d_μ threshold. The specificity of `NodeClustering` keeps on decreasing for the threshold values. The optimal value might be around the $0.6 \times d_\mu$, where all the known intrusions are reported as anomalies. At this point, about 80% of normal traffic is reported correctly by `NodeClustering`.

5.4.3 Comparisons with spectral clustering method

`NodeClustering` algorithm was applied on two network traffic datasets and results have showed the algorithm can detect all the known intrusions in data. Now we compare the results of this algorithm against two variants of a graph based spectral clustering method from Shi and Malik.

We compare results of `NodeClustering` with that of Shi and Malik's spectral clustering method called `normalized-cut` [32]. Shi and Malik perform generalized eigenvector decomposition between the un-normalized graph Laplacian and the node degree matrix, from which they use the eigenvector associated with the second smallest eigenvalue of the decomposition as graph partitioning criterion. This eigenvector which is associated with the second smallest eigenvalue is called Fiedler Vector.

We then follow two variants of the Shi-Malik spectral clustering comparison. We briefly list the algorithms here. The first method is the original method itself, shown in algorithm 5. We call this `Shi-Malik` 2.

Algorithm 5 Shi-Malik 2 [32]

1.Construct a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ from the data, where the weights (edge values) measure the similarity of the nodes
2. Solve the eigenvectors of $(\mathbf{D} - \mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$
3. Use the eigenvector associated with the second smallest eigenvalue to bipartition the graph
4. If necessary, partition the segmented parts recursively

In algorithm 5, there is possibility of recursive segmentation of the two partitions (from step 2) further. However, in anomaly detection, we need two groups showing normal and anomalous samples. Thus we do not perform step 4. Bi-partitioning of the graph is done by using the sign of the eigenvectors associated with the second smallest eigenvalue. Positive signed eigenvalues

make up one cluster and negative valued eigenvectors make another cluster.

The second method is a variant of algorithm 5, which we call `Shi-Malik 1` as listed in algorithm 6.

Algorithm 6 Shi-Malik 1 [6]

1. Construct a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ from the data, where the weights (edge values) measure the similarity of the nodes
2. Solve the eigenvectors of $(\mathbf{D} - \mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$
3. From the generalized eigen-decomposition of step 2, choose the first k-columns of the eigenvectors from decomposition in step 2
4. Perform k-means clustering on each row of the chosen eigenvectors

`Shi-Malik 1` uses k-means clustering with $k = 2$ on the eigenvector associated with the second smallest eigenvalue. We used the cluster indices as indicators of membership.

We used the mean of the node degrees as threshold of separation in algorithm 4.

The three algorithms were run on the same data segments that were drawn randomly from the datasets. This random splitting was done 30 times and average of the 30 runs was taken to get the sensitivity and specificity values of the algorithms. We will use these values as a measure of performance.

NSL-KDD 99 dataset

We randomly split the NSL-KDD 99 dataset into 30 sets. We used the scaling parameter of the RBF kernel as 0.1. We choose this value based on effect of the scaling on performance as shown in 5.10. We compare the performance of both implementations of Shi and Malik with our `NodeClustering` algorithm on the same data splits and we take the average of performance values on the runs. Table 5.3 shows the result of the comparisons on the performance of the three methods.

Table 5.3 shows `NodeClustering` method has the best sensitivity and the highest specificity values, which means that it detects all the known intrusions in the data and can detect 66.67% of the normal traffic as normal.

Method	Sensitivity	Specificity
`Shi-Malik 1`	0.5025	0.4962
`Shi-Malik 2`	0.5533	0.6598
`NodeClustering`	**0.9987**	**0.6670**

Table 5.3: Comparison of NodeClustering method on NSL-KDD 99 with two methods of Shi-Malik

Moreover, table 5.3 shows that `NodeClustering` has the highest true positive rate and lowest false positive rate. This shows that `NodeClustering` is very effective in retrieve intrusive nodes from the traffic data.

NG-Set

We follow the same procedure as the NSL-KDD 99 dataset. We choose the scaling parameter $\delta = 0.3$ since it had better performance on this scale. Figure 5.11 showed the plots, which depicts the effect of the scaling parameter on performance. Table 5.4 compares the sensitivity and specificity of the three methods.

Method	Sensitivity	Specificity
`Shi-Malik 1`	0.4891	0.5329
`Shi-Malik 2`	0.5145	0.4342
`NodeClustering`	**1.0000**	**0.6389**

Table 5.4: Performance of `NodeClustering` on NG-set with two methods of Shi-Malik

It can be seen `NodeClustering` has the best sensitivity and specificity. Similar to the NSL-KDD 99 dataset, `NodeClustering` detected all known intrusive samples and recognized 63.89% of the normal traffic as normal. Moreover, it has the lowest false positive rate.

From the above experiments on the two network traffic datasets, the method `NodeClustering` detected all known intrusive samples better than any of the Shi and Malik methods and has the smallest false positive rate.

5.4.4 Effect of normal traffic size

Until this point all the experiments had proportion of intrusive to normal traffic as 1 : 9. This made the experiments resemble a normally functioning network. We further check the validity of `NodeClustering` when the proportions change. We vary the normal traffic from 86% to 98% to emphasize different degrees of normality in the IP network. We test this effect in both datasets.

NSL-KDD 99 set

For threshold of normal and anomalous traffic samples, we randomly split the NSL-KDD 99 dataset 30 times and measure average of the specificity and sensitivity values of for each threshold and all the runs. We use the same evaluation matrix shown in table 5.1 to calculate the sensitivity and specificity values for the three algorithms.

Figures 5.13 and 5.14 show how the sensitivity and specificity values deviate with the amount of normal traffic. Node clustering method achieves the best sensitivity values for all the studies threshold limits. This proves again our method is superior and all the known attacks are identified by the method.

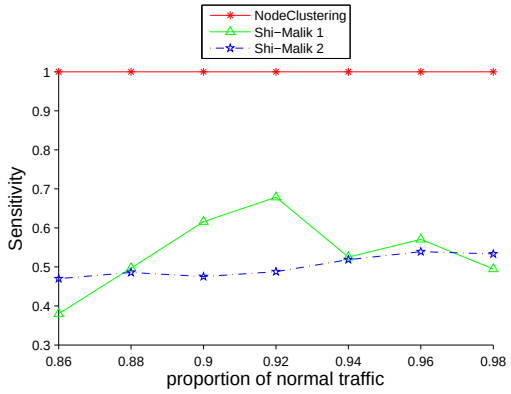

Figure 5.13: Sensitivity of NSL-KDD 99 dataset

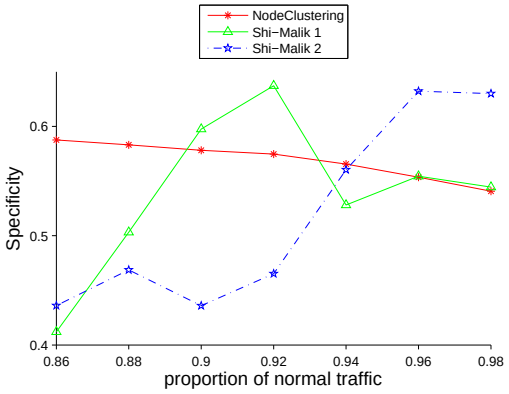

Figure 5.14: Specificity of NSL-KDD 99 dataset

In figure 5.14, the specificity values are indicated for the three methods.This value decreases when the proportion of normal traffic increases for the method `NodeClustering` whereas generally increases with increasing proportion of

normal traffic data size in `Shi-Malik 2`. However, Specificity of `Shi-Malik 1` increases in the beginning before starting to decline.

In table 5.1, we showed false positive rate is one minus the specificity. Increasing specificity means decreasing the false positive rate. Thus, `Shi-Malik 2` has better better false positive rate with increasing normality in the data although the results have some variances and `NodeClustering` has results with less variance.

NG-set

We follow similar procedures in the NG-set as in the NSL-KDD 99 set. We make 30 experimental runs for 30 random splits of the NG-set with nine different normal and anomalous sample thresholds. We vary the amount of normal traffic from 86% to 98%. In addition, the scaling parameter of the RBF kernel was set as 0.3

Figures 5.15 and 5.16 show the sensitivity and specificity of the three methods. Figure 5.15 visualizes that when the amount of normal traffic is around 86%, the `NodeClustering` algorithm performs around 20% less than both implementations of Shi and Malik. When the amount is increased to 98%, `NodeClustering` is able to detect all the attacks in the data and outperforms both implementations of Shi and Malik with an improvement of almost 50% true positive rate. Their method finds only about 50% of the attacks in the data, while `NodeClustering` finds almost all the intrusive samples as attacks.

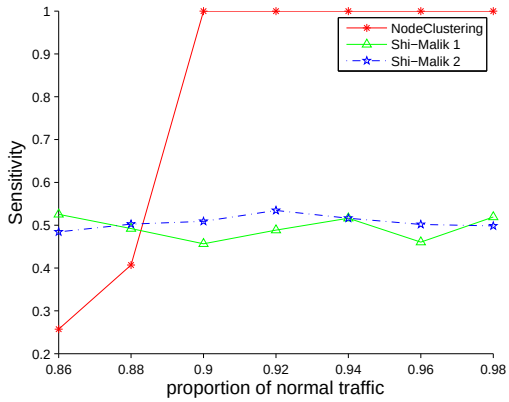

Figure 5.15: Sensitivity of NG-set with

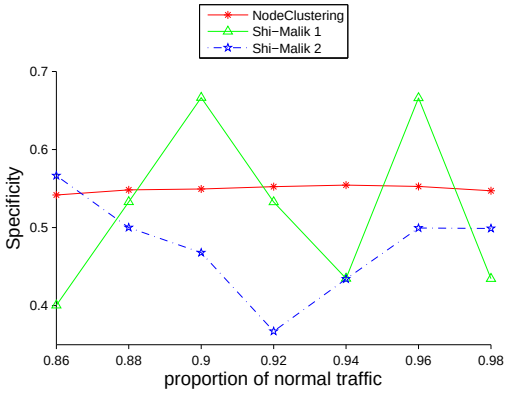

Figure 5.16: Specificity of NG-set

Figure 5.16 plots the specificity values. These values are related to the false positives rates. Our node clustering algorithm has almost stable specificity of about 55%, which is equivalent to 45% false positive rate for most of the normal traffic thresholds. This means `NodeClustering` finds around 55% normal traffic from the total traffic normal set. The two implementations of Shi and Malik have unstable ranges of specificity values for the different amount of traffic. `Shi-Malik 1` generally gave a better specificity value with increasing size of normal traffic than `Shi-Malik 1`.

The comparisons in this subsection showed that `NodeClustering` is a very effective method in retrieving intrusive traffic, which are subsets of anomalies. In normal operation of a network, there are few anomalies (e.g. attacks) and majority of the subsets in the traffic data are normal samples. The experiments with different sizes of normal traffic have shown that `NodeClustering` gives the best true positive rate especially in a normally operating networks.

5.4.5 Application to other datasets

In this section, the validity of `NodeClustering` to non-security related datasets is shown. The reason is to demonstrate the robustness of the method and that it can be used to study similar group of samples in any data. However, it should be noted that as `NodeClustering` is a bi-partitioning method and find two different clusters.

Fisher Iris dataset

Fisher iris data has well known classes (clusters) is studied. A general fact about this dataset has three classes. The setosa class includes all the samples 1 to 50, versicolor class from sample 51 to 100, and virginica class from sample 101 to 150. And the discovered groups coincided with the structure defined by the true labels.

Weighted undirected graphs were constructed in similarly manner with the traffic datasets. Node degree is computed from the adjacency matrix of the graph to find similar nodes in the graph.

Figure 5.17 plots of the node degree of the iris graph. It shows that there is one group on the lower left corner until the 50-th sample point, followed by a long jump into another disperse group on the right top corner which plots the rest of the 100 samples. In this left group, the nodes show scattered distribution which looks difficult to be categorize as one.

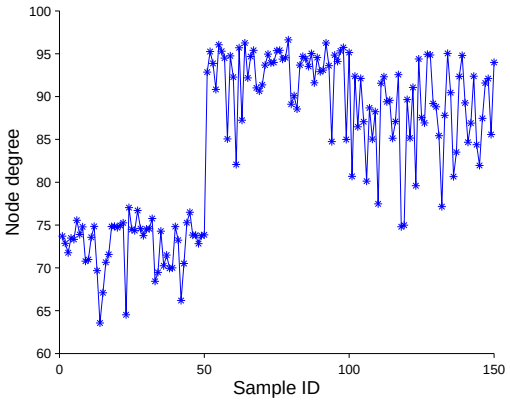

Figure 5.17: Node degree plot for Fisher Iris data

The above problem showed that two groups exist with one of the groups showing un-generalized behavior and other group on the left bottom corner is clearly visible. We further investigate the nodes in both regions using the transition probabilities of $\mathbf{P}$ matrix and picking nodes from the regions.

Figures 5.18 and 5.19 plot the transition matrix entries $\mathbf{P}(\mathbf{1}, \cdot)$ and $\mathbf{P}(\mathbf{150}, \cdot)$. From these plots, it can be shown that there are three groups with the transition occurring at the 50-*th* and 100-*th*.

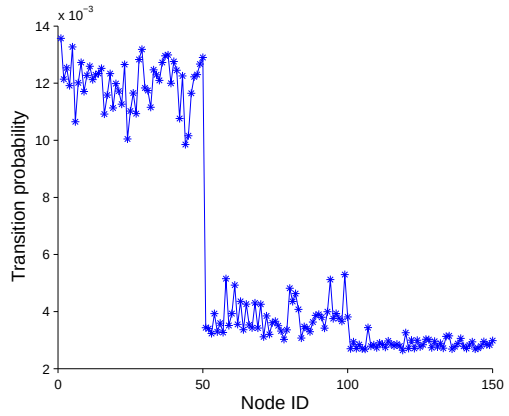

Figure 5.18: Fisher Iris data transition probability of node 1

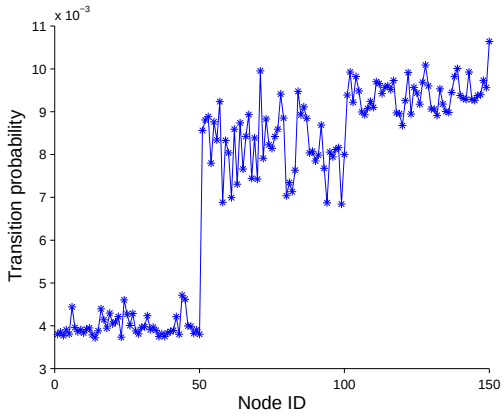

Figure 5.19: Iris data transition probability of node 150

When applying node clustering algorithm, it will retrieve the first class in the iris data as one cluster, and the other two classes as another cluster. This shows that node clustering will effectively work as a graph bi-partitioning method when there are two clearly distinct classes, but when there are more classes it will report two of the most distant clusters.

5.5 Summary

The study on two network traffic and one non-network traffic sets indicated that applying `NodeClustering` on graphs from these sets will bipartition the nodes. This will give two sets of nodes that are more similar to each other. We effectively recovered all the known intrusive samples using the node degree values. Results have been exemplified with visualization of the node degree values and their Markov random field transition probabilities. In addition, we observed the following results with the experiments:

- `NodeClustering` retrieved all the intrusive samples almost with 100% true positive rate
- how the results of `NodeClustering` change with the scaling of the kernel function
- how `NodeClustering` performs in different IP networks by varying the amount of normal and anomalous samples in the traffic sets and how it performed better in highly normal operating IP networks, making it realistic choice

- how the threshold of separation into normal and anomalous samples affects the result of `NodeClustering`
- `NodeClustering` outperformed two variants of Shi and Malik methods and results have been compared for different size of normal traffic data
- the performance of `NodeClustering` with more than two classes (clusters).

Chapter 6

Conclusions

The thesis studied the problem of anomaly detection in IP networks. Anomaly detection identifies different events that generate abnormal traffic patterns, for example, attacks or unknown rare events in networks. The goal was to find anomalies in network traffic datasets collected from normally operating IP networks.

A new graph based clustering algorithm, called `NodeClustering`, was designed. This method successfully identified all the known intrusive samples, which are one example of anomalies.

`NodeClustering` has a computationally smaller running time compared with spectral clustering methods, which is only the time it takes to construct the graph. This property is useful in live monitoring of networks. Other spectral clustering methods reported half the known intrusive samples as anomalies with longer computational time while `NodeClustering` took smaller running times with better true positive rates. The spectral clustering algorithms for instance, Shi-Malik's method focus on finding two comparably sized clusters while `NodeClustering` is effective in situations where there is one large cluster, containing majority of the samples, and few points that are distant from this cluster.

Moreover, `NodeClustering` method provides system administrators and security analysts a way to identify attacks, device failures and other rare unknown events. This makes `NodeClustering` method an important method for anomaly detection and troubleshooting of failures in the network. It uses the node degree property of the graph to partition the nodes without com-

puting the graph Laplacian or its spectral decomposition. Moreover, the criterion for partitioning is simple and heuristically motivated. In addition, `NodeClustering` was efficient in identifying intrusions in a normal operation of a network as shown in the experiments.

Currently, `NodeClustering` algorithm bi-partitions graphs into two sets of nodes. Although setting a threshold is difficult, in the future `NodeClustering` can be optimized to find even more clusters using a different criteria for partitioning. This might result in a much smaller false positive rate while maintaining the current true positive rate.

Additional investigation can be done in scaling the number of samples the graph construction handles. Moreover, future work also will be constructing models for normal and anomalous graphs and comparing new graphs based on their degrees to these reference graphs by using some distance measures like graph edit distances and also how to integrate discrete graphs into the anomaly detection and how to use the `NodeClustering` for on-line learning.

Bibliography

[1] James P. Anderson Co, Computer Security Threat Monitoring and Surveillance, Technical report,Fort Washington,Pennsylvania, April 1980.

[2] Karen Scarfone and Peter Mell, Guide to Intrusion Detection and Prevention Systems (IDPS), Recommendations of the National Institute of Standards and Technology,2007.

[3] Rebecca Bace and Peter Mell, Special Publication on Intrusion Detection Systems, Recommendations of the National Institute of Standards and Technology (NIST), 2001.

[4] Aurobindo Sundaram, An introduction to intrusion detection, Special issue on Computer Security, Crossroads 2(4)(1996) 3-7.

[5] Pedro Garcia-Teodoro and Jesús E. Díaz-Verdejo and Gabriel Maciá-Fernández and Enrique Vázquez, Anomaly-based network intrusion detection: Techniques, systems and challenges, Computers and Security 28, 1-2 (2009) 18-28.

[6] Ulrike von Luxburg, A tutorial on Spectral Clustering, Statistics and Computing 17(4)(2007) 395-416.

[7] Matthias Hein, Jean-Yves Audibert and Ulrike von Luxburg, Graph Laplacians and their Convergence on Random Neighborhood Graphs, Journal of Machine Learning Research 8 (2007) 1325-1368.

[8] S.V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor and Karsten M. Borgwardt, Graph Kernels, Journal of Machine Learning Research 11 (2010) 1201-1242.

[9] Bernard Schölkof and Alexander J. Smola, Learning with Kernels, The MIT Press, Cambridge 2002.

[10] Michel Neuhaus and Horst Bunke, Bridging the Gap Between Graph Edit Distance and Kernel Machines, Series in Machine Perception and Artifical Intelligence, 2007.

[11] Xiaojin Zhu, Zoubin Ghahramani and John Lafferty, Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions, In *The 20th International Conference on Machine Learning (ICML)*, 2003.

[12] Christophe Rosenberger and Luc Brun, Similarity-based matching for face authentication, In *The 19th International Conference on Pattern Recognition (ICPR)*, 2008.

[13] Satu Elisa Schaeffer, Graph Clustering,Computer Science Review 1(1) (2007) 27-64.

[14] S. Boccaletti, V. Latora, Y.Moreno, M.Chavez and D.-U. Hwang, Complex networks: Structure and dynamics, Physics Reports 424 (2006) 175-308.

[15] Marina Meila and Jianbo Shi, Learning Segmentation by Random Walks, In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[16] Lovász, Random walks on graphs: a survey, In Combinatorics, *Paul Erdös is eighty*, Budapest: János Bolya Math. Soc, 1993.

[17] Fan Chung, Spectral Graph Theory, CBMS Regional Conference Series in Mathematics, No. 92, 1997.

[18] Michael Newman, Laplacian Spectrum of Graphs, PHD Thesis, 2000.

[19] Huaijun Qiu and Edwin R. Hancock, Graph matching and clustering using spectral partitions,Pattern Recognition 39(1) (2006) 22-34.

[20] Tore Opsahl,Filip Agneessens and Jhon Skvoretz, Node centrality in weighted networks: Generalizing degree and shortest paths,Social Networks 32(4) (2010) 245-251.

[21] Ernesto Estrada and Naomichi Hatano,A Vibrational Approach to Node Centrality and Vulnerability in Complex Networks,Physica A 389 (2010) 3648-3660.

[22] Roberto Navigli and Mirella Lapata, An Experimental Study of Graph Connectivity for Unsupervised Word Sense Disambiguation, IEEE transaction on pattern analysis and machine intelligence 32(4) (2010) 678-692.

[23] Ulrik Brandes, A Faster Algorithm for Betweenness centrality Journal of Mathematical Sociology 25 (2001) 163-177.

[24] Kazuya Okamoto, Wei Chen and Xiang-Yang Li, Ranking of Closeness Centrality for Large-Scale Social Networks, Frontiers in Algorithmics 5059 (2008) 186-195.

[25] Iacopo Carreras, Daniel Miorandi, Geoffrey S. Canright and Kenth Engo-Monsen Eigenvector Centrality in Highly Partitioned Mobile Networks: Principles and Applications,Studies in Computational Intelligence(SCI), 69 (2009) 125-147.

[26] Piet Van Mieghem, Graph Spectra for Complex Networks, Cambridge University Press,January 2011.

[27] Google Home Page `http://www.google.com`. Accessed 16 September 2011.

[28] Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, In *Proceedings of the Seventh International World Wide Web Conference*,1998.

[29] C Bishop, Pattern Recognition and Machine Learning. Springer, Massachusetts, 3rd Edition,2006.

[30] A.K. Jain, M.N. Murty, P.J Flynn, Data Clustering: A Review, Computing Surveys 31(3) (1999) 264-323.

[31] Ethem Alpaydin,Introduction to Machine Learning, Second Edition,The MIT Press,2010.

[32] Jianbo Shi and Jitendra Malik, Normalized Cuts and Image Segmentation IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8)(2000) 888-905.

[33] Francis R.Bach and Michael I.Jordan, Learning Spectral Clustering,2003.

[34] Andrew Y. Ng, Michael I. Jordan and Yair Weiss, On Spectral Clustering: Analysis and an algorithm, In *Advances in neural information processing systems (NIPS)*, 2001.

[35] Dhillon, Inderjit S. and Guan, Yuqiang and Kulis, Brian, Kernel K-means, Spectral Clustering and Normalized Cuts, In *Proceedings of the tenth ACM SIGKDD International conference on Knowledge discovery and data mining (KDD'04)*,(2004) 551-556.

[36] C.Ding and X. He. Linearized Cluster Assignment via Spectral Ordering, In *proceedings of International Conference on Machine Learning*, New York 2004.

[37] Pekka Kumpulainen and Kimmo Hätönen, Local Anomaly Detection for Mobile Network Monitoring, Journal of Information Science 178(20) (2008) 3840-3859.

[38] The kDD-CUP-99 task description, `http://kdd.ics.uci.edu/databases/kddcup99/task.html`, Accessed 11 May 2011.

[39] DARPA Intrusion detection dataset, Lincoln Laboratory, Massachusetts Institute of Technology, `http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html`, Accessed 2 August 2011.

[40] Lincoln Laboratory, Massachusetts Institute of Technology, `http://www.ll.mit.edu/l`, Accessed 2 August 2011.

[41] Argus-Auditing Network Activity, `http://www.qosient.com/argus/index.shtml`, Accessed 13 July 2011.

[42] Argus-Manual,`http://www.qosient.com/argus/manuals.shtml`, Accessed, 13 July 2011.

[43] Jhon McHugh, Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory, ACM Transactions on Information and System Security 3(4) (2000) 262-294 .

[44] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani, A Detailed Analysis of the KDD CUP 99 Data Set, In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA)*, 2009.

[45] The NSL-KDD Data Set, `http://iscx.ca/NSL-KDD/`. Accessed 3 June 2011.

[46] Metasploit Framework, Penetration Testing Software `http://www.metasploit.com/`. Accessed 3 August 2011.

Appendix A

Feature Description of the NSL-KDD 99 set

Feature name	feature description
duration	length of the connection
src-bytes	number (num) of bytes from the source to the destination
dst-bytes	num of bytes from the destination to the source
wrong-fragment	num of wrong format
urgent	num of urgent package
hot	num of "hot" indicators
num-failed-logins	num of failed attempts to log-in
num-compromised	num of compromised conditions
num-root	num of root access
num-file-creations	num of file creation operations
num-shells	num of shell prompts
num-access-files	num of operation performed on access control files
num-outbound-cmds	num of outbound commands for `ftp` session
count	num of connection to same host during last 2 seconds (sec)
srv-count	num of services to the current service during last 2 secs
dst-host-srv-rerror-rate	% of connections with `REJ` rate between destination and host with same srv
serror-rate	% of connections with `SYN` error
srv-serror-rate	% of connections (con.) with `REJ` with same service
rerror-rate	% of con. with `REJ` error
Continued on next page	

Table A.1 – continued from previous page

Feature name	feature description
srv-rerror-rate	% of con. with `REJ` with same service
same-srv-rate	% of con. with same service rate
diff-srv-rate	% of con. with different service(srv) rate
srv-diff-host-rate	% of con. to different hosts
dst-host-count	num of con. between a destination(dst) and host
dst-host-srv-count	num of con. between dst and host with same service
dst-host-same-srv-rate	% of con. between dst and host with same service
dst-host-diff-srv-rate	% of con. between dst and host with different service
dst-host-same-src-port-rate	% of con. between dst and host with same source port
dst-host-srv-diff-host-rate	% of con. between dst and host with different host and srv
dst-host-serror-rate	% of con. with `SYN` rate between destination and host
dst-host-srv-serror-rate	% of con. with `SYN` rate between dst and host with same srv
dst-host-rerror-rate	% of con. with `REJ` rate between dst and host with same srv

Table A.1: Continuous featues of the NSL-KDD 99 dataset [38]

Appendix B

Feature Description of the Ng-dataset

Feature	**feature description**	**type**
dur	total duration of the record	continuous
stime	start time of record	continuous
ltime	ending time of record	continuous
mean	average duration for aggregated records	continuous
sttl	time to live values from source to destination	continuous
proto	type of protocol in the transaction	discrete
pckts	Total count of packets in the transaction	continuous
saddr	source IP address	discrete
sport	source port number used	continuous
spkts	count of packets sent from the source to destination	continuous
sbytes	transaction bytes from source to destination	continuous
daddr	destination IP address	discrete
dport	destination port number	continuous
dpkts	count of packets from destination to the source IP address	continuous
dbytes	transaction bytes from destination to source	continuous
load	bits per second	continuous
sload	source bits per second	continuous
dload	destination bits per second	continuous
loss	packets dropped or retransmitted	continuous
sloss	source packets dropped or retransmitted	continuous
dloss	destination packets dropped or retransmitted	continuous

Table B.1: Features of the Ng-set

Appendix C

Feature Extraction with Argus

The argus command converts the captured PCAP file format into an argus readable format.

```
argus  -r  TrafficWithAttacks11.pcap  -w  file1.argus
argus  -r  TrafficWithAttacks12.pcap  -w  file2.argus
```

The actual feature extraction is specified in the command line for the read argus tool (**ra**) shown below. Each of the features are described in the table in Appendix A above.

```
ra - u -nr  file1.argus -c ";" -s dur stime ltime mean sttl
proto pkts saddr sport spkts sbytes daddr dport dpkts dbytes load
sload dload loss sloss dloss ploss psloss > NgSet1.txt
```

```
ra - u -nr  file2.argus -c ";" -s dur stime ltime mean sttl
proto pkts saddr sport spkts sbytes daddr dport dpkts dbytes load
sload dload loss sloss dloss ploss psloss > NgSet2.txt
```