Kernighan-Lin Heuristic in an IDS

Mats Erik Smestad

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2006

Abstract

With the need for organizations to stay online and connected to the World Wide Web, the need for security measures is stronger than ever. One important security feature is the utilization of an Intrusion Detection System, IDS. But as this is a relatively new technology it is still hazed in a grey cloud of confuse and misunderstanding. This thesis will focus on giving the reader an insight of IDS and methods used. The Intrusion Detection Systems of today have yet to prove their full potential. An especially strong feature of an IDS is to alert currently unknown attacks/misuse as it detects traffic that is outside the normal boundaries. Unfortunately this kind of IDS have a tendency to produce a high number of alerts on normal traffic as well. This thesis will investigate if the Kernighan-Lin algorithm may be used in an Intrusion Detection System and compare the results with the k-means algorithm. The Kernighan-Lin has not been used in an IDS, but k-means have already been implemented and tested. The study of Kernighan-Lin heuristic will hopefully reveal if it can perform better than the k-means regarding accuracy without a significant loss in speed.

The testing of the two algorithms, Kernighan-Lin and k-means, shows that Kernighan-Lin has a good potential as a classifier in an IDS. It does have some limitations, but they can be solved or circumvented. The true strength of Kernighan-Lin was the accuracy it provided. It managed to correctly identify each attack. However the tests done do favor the Kernighan-Lin in some degree.

Sammendrag

Organisasjoner og bedrifter i dag krever å være online og koblet til Internet. Dette stiller høye sikkerhetskrav. I perimetersikring for organisasjonen er Intrusion Detection System, IDS, et viktig element. Men dette er en ung teknologi som på grunn av manglende forståelse og kunnskap ikke blir benyttet i særlig grad. Denne Masteroppgaven tar for seg en forklaring av IDS og kort forklaring om de forskjellige måtene som en IDS opererer på. Intrusion Detection Systemer har enda å vise sitt fulle potensiale som et forsvarsverk. En veldig sterk egenskap er at en IDS kan oppdage angrep før de er kjent, ved at den sammenligner normal trafikk mot unormal trafikk. Desverre har denne typen IDS en tendens til å generere mange falske advarsler. Denne Master0ppgaven skal undersøke om det er mulig å forbedre feilraten i en IDS ved å benytte Kernighan-Lin heuristic og sammenligne de resultatene med k-means heuristic. Kernighan-Lin er ikke benyttet i en IDS, mens k-means har allerede blitt implementert og testet.

Testing av de to algoritmene, Kernighan-Lin og k-means viste at Kernighan-Lin har et godt potensiale i en IDS. Den har noen begrensninger men disse kan løses. Styrken til Kernighan-Lin ligger i at den er veldig presis til å identifisere angrep. Forsøkene gjort i denne Masteroppgaven viser at Kernighan-Lin er veldig presis, men den har også en fordel i forhold til k-means testen.

Contents

List of Tables

List of Figures

List of Definitions

Algorithm: A step by step procedure for solving a problem.

A problem: A question that must be answered.

Accuracy: False positive / true positive ratio.

Node: One element in a given graph.

Feature: A coordinate of a vector.

Vector: Distant measure to a node.

Graph: A collection of nodes with a connection to one another.

Edge: The connection between nodes. This connection may have a specified cost which describes the cost for traveling to one node to another.

False Positive: A measure of error of the conclusion of assumed attacks/non-attacks given by an IDS and the actual attacks in the dataset.

True positive: A measure of correctly assumed attacks/non-attacks made by the IDS.

Cluster: A specific group of features arranged by an algorithm often determined by similarities of a given attribute.

IDS: Intrusion Detection System. It is the totality of a defense mechanism design to act like a burglar alarm. This is often in order to detect potential attack/misuse in an organization's network.

Misuse: Undesired traffic in a network. This may be a direct attack, malware, worms, trojan, spyware, virus or unintentional usage of resource by either an insider or outsider.

Anomaly: A pattern that is outside the normal. It might for example be malicious traffic as this has characteristics that separate it from normal traffic.

Summary of Results

This thesis has demonstrated that one can use Kernighan-Lin clustering algorithm for classification of network patterns in an Intrusion Detection System, and that this algorithm performs very well.

The Kernighan-Lin heuristic partitioning was tested with regards to the well known k-means algorithm in an Intrusion Detection System. Both were tested on the same computer with the same dataset and the results of the Kernighan-Lin are more accurate then those attained by means of the k-means algorithm.

The results were based on the false positive / true positive ratio. The dataset KDD-Cup was used as the number of attacks within this set was known.

The experiment was designed to answer the following questions:

“Will the Kernigan-Lin Heuristic produce a higher or lower true / false positive rate than the k–means partition?”

The hypothesis was that Kernighan-Lin will have a lower false positive rate than the k-means. The experiment in this thesis confirms this hypothesis. And the second hypothesis was that Kernighan-Lin would be slower. This was also confirmed, but there is a solution to this and an IDS can still run efficiently with the Kernighan-Lin heuristic implemented.

Acknowledgments

I have been very fortunate to receive very good help throughout the whole thesis. I would like to thank prof. Slobodan Petrovic for supervising this thesis, both in the planning phase as well as during the experiment. He has been a very good support in the coding phase and provided many good ideas for this thesis. I would also thank both previous and recent master students for helpful input and comments during the writing of this thesis.

A special thanks goes to my friends and family for being supportive and patient during the writing of this thesis.

Preface

Mats Erik Smestad has a bachelor degree in computer science in Gjøvik University College, Norway. Currently he studies Masters in Information Security at the same University College. This Thesis will conclude the Masters Program education from autumn 2004 to spring 2006.

The main reason for choosing this topic was based on my knowledge and interest in Intrusion Detection Systems. I also wanted to use my skills as a programmer and I find this topic highly relevant in years to come. The need for even better security is more relevant now than ever and it is a trend that seems to be rising. IDS is a young technology that requires more research and development. This thesis investigates an algorithm that has potential to improve current Intrusion Detection Systems.

Mats Erik Smestad, 2006/06/01

1 Introduction

1.1 Topic

As more and more organizations rely on Internet the need for security is increased. The number of attacks on the World Wide Web is getting larger and the methods used are sophisticated. To protect themselves, organizations make use of different security systems and one of those are Intrusion Detection Systems, IDS. IDS is a relatively young technology and have yet to reach common acceptance. One of the shortfallings is that in order to alert about yet unknown attack one might be overwhelmed with false alarms. In other words, the system triggers an alarm on traffic that is normal and not an attack. There is a need to improve the IDS so that false alarms are at a minimum with as small as possible loss of speed. This thesis investigates if it is possible to use the Kernighan-Lin heuristic instead of the k-means heuristic and thus produce better results regarding false positive / true positive ratio in an Intrusion Detection System. A description of clustering and Intrusion Detection are presented as well as an introduction to the algorithms k-means and Kernighan-Lin.

Keywords: Intrusion Detection System, clustering, patterns, k-means algorithm, Kernighan-Lin algorithm.

1.2 Problem Description

Most Intrusion Detection Systems today rely on signature based detection. The disadvantage with these systems is that they can only detect known attacks. The introduction of anomaly based Intrusion Detection Systems shows that it is possible to create a detection system that also detects attack that are currently unknown. Such systems are still at an early stage of development, but as new attacks emerge almost every day, the need for such intrusion detection is present. To separate misuse from normal attack may prove very difficult as they tend to overlap each other. In many cases there is no clearly defined distinction between the two. It is this overlap that causes the false positive and false negative results, and developers strive to increase the accuracy of Intrusion Detection Systems. Algorithms are improved and methods are researched to improve the performance of an Intrusion Detection System. Two imperative elements are speed and accuracy; the IDS must give as accurate results as possible within as short time frame as possible. The k-means is a well-known algorithm for use within an Intrusion Detection System because of its simplicity and speed, but it does have its flaws. It may cause high false positive rate in the overall IDS.

There are several ways to improve an Intrusion Detection System. One might try to further improve the algorithm, test other algorithms or look at a new method. And for this thesis the focus has come to a new algorithm that has yet to be tested in an Intrusion Detection System. This algorithm, Kernighan-Lin, has proved itself in other areas [2] and it is interesting to see how the performance is, compared to k-means regarding both accuracy and speed.

1.3 Justification, Motivation and Benefits

The extra security an Intrusion Detection System might add is valuable for many organizations. To develop an application or configure a system to be absolutely secure is extremely difficult. And it can be expensive to replace existing systems with more secure ones [32]. An IDS can aid the security of a system as it may detect attacks and misuse that goes unnoticed by a firewall or anti virus software. The real strength and potential of an Intrusion Detection System based on anomaly detection is that it can detect an unknown attack. This does not apply for a signature based IDS. However the problem for an anomaly based IDS is the accuracy as the false positive rate is too high. As most Intrusion Detection Systems today rely on signature based detection, it does not differ too much for other well-known security measures. Furthermore, IDS suffers with high complexity and high price so many organizations use other means to secure themselves. And often the vulnerability is ignored as a whole.

In security it might prove very valuable to be proactive instead of reactive. If a brand new attack occurs, it would benefit greatly to be ahead of the attack. Most systems today are reactive in that they rely on signatures; they search data for known features. But then the attack is already known and may have crippled the system or sensitive data may already been stolen. It is desired to have a high probability to be warned about every type of attacks, new or old, as they are used on the system; this is a proactive approach. This approach may also detect unintended misuse that was not foreseen when the system was designed. An anomaly Intrusion Detection System is proactive, but unfortunately it suffers from a high degree of false alerts, or false positives. It tends to generate a warning when there in fact are no attacks.

In this thesis looks deeper into the fact that anomaly based Intrusion Detection System have too high false positive rate. This have a very negative side effect in that people tend to loose faith in the system and the IDS just becomes a big investment for the organization without any gain. If the false positive rate would be greatly lowered more people could gain confidence in the system.

1.3.1 Benefactors

Stakeholders would be IDS operators, computer security managers in general and researchers that try to develop more reliable anomaly based intrusion detection. If it is proven that Kernighan-Lin does have a much higher accuracy without great loss of speed this will help further researcher on developing a better and more accurate IDS.

1.4 Research Question

To be able to test the two algorithms as accurately as possible it is chosen to use KDD-Cup data set as test set. This has the advantage of knowledge of how many attacks/misuse are present in the dataset and will give a good measure of correctness of each algorithm. There are several algorithms to be tested but this thesis will concentrate on two algorithms. The first one is the k-means, an algorithm used in Intrusion Detection System, and the Kernighan-Lin. The latter algorithm has yet to be tested in an Intrusion Detection System.

This leads to the following research questions:

1. To what degree can Kernighan-Lin Algorithm be used in an Intrusion Detection System for anomaly detection?

2. Analyze the k-means vs. Kernighan-Lin algorithms and see whether Kernighan-Lin has a higher or lower false positive rate than k-means.

Hypothesis: The Kernighan-Lin will provide a lower false positive rate than k-means.

3. Is there a significant speed difference between the two systems?

Hypothesis: Kernighan-Lin will be slower than k-means, but workaround is possible.

2 A Review of State of the Art

The first Intrusion Detection System was developed as an automated method for reviewing audit trails [3]. The system had a goal to remove redundant work and make it easier to focus on the log entries that needed attention. In mid 80s Dorothy Denning and Peter Naumann developed a real-time IDS and this model tried to discern attacks by anomaly detection. In this period several IDS system were born; IDES, Discovery, Haystack, MIDAS and some others. Most of these systems were designed for military use. The internet had yet to become the Net we know today, so commercial use was limited to the major banks and the largest of corporations [23]. Research and development of Intrusion Detection System is relatively young [3] [12]. It has been studied for approximately 30 years, and given the complexity of the problem the IDS is still at an early stage. Several problems have to be worked out before a really good IDS may be presented [10]. Often they generate a high degree of false alarms [21]. Most commercial and available Intrusion Detection Systems today are misuse based, or signature based. The signature based IDS look for distinct features and see if they match a predefined database. This database holds data that describes what attack data looks like, the attack-data’s signature. This is the easiest IDS to build and as a general rule it generates less False Positive results. But they also tend to miss more attacks as they slip past the system unnoticed.

2.1 Anomaly

Anomaly based Intrusion Detection Systems separate malicious traffic from normal traffic by other means. They can for instance use learning or clustering [1]. When learning is used the system must be supervised with a manager that tells the system what it should react on. This manager uses a sample dataset, often artificially generated, and as the system responds the manager will ‘tell’ whether the system acted correctly or not. This is a tedious task and often leads to wrongly configured systems as the dataset may not be proper for the given domain it should represent. The difficulties of making a good initial dataset and the time it takes to teach the IDS how to act makes learning based IDS expensive and complex. But it is still early in the development and learning may soon get the necessary progress to become a reliable Intrusion Detection System.

2.2 Clustering

Clustering is perhaps the method with the most promising result. Clustering is often referred to as unsupervised classification. It does not require an initial dataset and the strength of the system lies within the algorithm itself. Several algorithms have been proposed [1]. Single-Link Clustering algorithm, k-means (Squared error Clustering), hierarchical clustering algorithm to mention a few [1] [16] [24] [30]. This thesis takes a look at a clustering algorithm called Kernighan-Lin. This is a well known algorithm for partitioning nodes of a graph and is much used as a tool for assigning components of electrical circuits on circuit boards [2] [7]. This algorithm has been proved to be very good at finding close to the optimal partition in relatively short time. However this algorithm has not yet been tested as a classification module for an IDS. Many experiments are conducted with different types of cost matrices, both in 1-0 matrices and integer matrices [2]. But not so much research has been done at float type matrices. With the given dataset, KDD-Cup, and Euclidean measure we would get decimal number as vectors. This thesis researches if the Kernighan-Lin algorithm can be used as a categorizer between attacks and normal traffic in an IDS.

3 Claimed contribution

The primary goal of this thesis is to research possible improvements of the results in an Intrusion Detection System that uses clustering techniques. In the search for increased accuracy in anomaly based intrusion detection it was interesting to take a look at the Kernighan-Lin algorithm and use that instead of the k-means. The focal point was to research the false positive rate of the known algorithm, the k-means, and see if the Kernighan-Lin may have the potential to replace it. The contribution consists of studying if the Kernighan-Lin produces a better accuracy than k-means. It may seem that Kernighan-Lin is a lot slower algorithm but this thesis briefly takes a look if this decreased speed is of greater significance. This is achieved by realizing the Kernighan-Lin and testing this within an IDS.

4 About Clustering and IDS

4.1 IDS

An IDS, or Intrusion Detection System, is basically a burglar system for a network or computer system. One general definition is: “It is defined as the process of monitoring the events occurring in a computer system or network, analyzing them for signs of security problems” [3]. Security problems are defined by the system’s security polices at each place and may greatly differ from system to system. An Intrusion Detection System may greatly strengthen the defense of a system [23]. It is an extra security perimeter behind the firewall, or often behind the firewall, and if properly installed it enhances the overall security of the network. Although it is a young security system of which much more research is needed [3] many good results have been achieved.

It has been shown that comparing to standard manual log reading by humans Intrusion Detection Systems are far more accurate. In an early study of Intrusion Detection Systems, a prototype was tested and it detected several hundreds of attacks, and manually log reading by humans only detected about 1-2% of these attacks [3]. So the system proves that an automated burglar system is far more accurate, it detects more attacks, than manual log reading. It’s also a lot faster and can be performed at a near real-time basis. It has the capability to detect brand new attacks without any prior knowledge, this type of IDS is called anomaly based IDS. The other type is signature based which is faster but cannot detect new unknown attacks.

The task of an Intrusion Detection System is to alert the manager of the network when an incident has happened. This incident may be an attack from the outside. For instance an attacker that tries to break into the network and steal valuable data. Even though he bypasses the firewall, and when he does the firewall probably won’t bother him anymore, the Intrusion Detection system scans the traffic all the time and when a particular threshold is reached an alert is generated and sent to the manager of the network [25]. The same thing applies when an attack comes from the inside. It monitors the entire network and logs so that even if the attacker is behind the firewall at all times the Intrusion Detection System can detect the attack.

In addition the IDS may detect unintentional errors from users and generate an alert before some critical errors actually occur. The IDS itself does nothing except monitor, detect and then generate an alert. The executive step, or the counter action to a misuse/anomaly, is done by the system manager. A further development, or an enhancement, of the IDS results in an IPS.

4.1.1 IPS

IPS is an intrusion Prevention System, and as such it does the same as IDS but additionally it has an executive step so when a misuse/anomaly is detected it may apply some countermeasure to limit the damage. An IPS is popularly said to be a combination of an IDS and a Firewall. What the IPS often does when an attack is detected is to block the invading source, be it from the inside or outside. This block may be permanent until a manager lifts it or it may be a temporary block that lasts just a given period of time before it is automatically lifted. But as mentioned before the IDS is not yet a perfect, or even near perfect system, so letting the system decide the proper countermeasure on its own may be as fatal as the attack itself.

4.1.2 Identifying attacks

The IDS have many challenges to be addressed and solved. The primary problem is the difficulty to separate normal traffic from the actual attack as they seem to blend together. This issue is somewhat taken care of in a *signature based IDS*, but also this system may fail in discerning attacks from normal traffic. The signature based IDS have a database in which special features uniquely identify a particular attack, much like a traditional antivirus program. It has to be updated regularly to detect new attacks and its potential to detect attacks that yet are to be discovered is greatly reduced [10]. The database may get very large and prognoses show that it will only get larger as time goes by. The larger the database the slower the system will be. Also the manager of the system must carefully look through every scenario to predict potential misuse, intended or unintended, to create a functional IDS [25].

The other type of IDS that try to combat this is an *anomaly based IDS*. This system has a particular algorithm, or several algorithms, that separate data by looking at the features, and in this way tries to find the attack among the data. It does not have a database on which it relies, but the algorithm looks at a portion of the data and analyses this. This type of IDS will be discussed further in this chapter.

The current products offering today are not good in any aspect [10]. The biggest problems are false alarms given when the system thinks it is an attack but the fact is that it is normal traffic. When systems are designed to produce less false alarms it tend to detect less malicious traffic as false positive and false negative are closely related [23]. Slow attacks may also present a problem for the IDS especially if it is to run in real time.

The IDS also have to combat the time aspect. The data gathered in networks today are quite a large set and to process all this is time consuming. It was questioned earlier if IDS had to define a new time definition known as near real time. The time between the data is gathered and then processed before a result is generated is not in real time. But given the data power and the strength of the algorithms today this issue is no longer a concern [3] [24].

4.1.3 Security Policy

The IDS can never be better than the security policy. A poorly designed security policy will also effect the configuration of the IDS. A security policy is often derived from the three basics of information security [11]:

- Confidentiality: Make sure that only authorized persons get access to classified information and that proper measure for identifying and authorization are met.
- Integrity: The information remains intact, accurate, valid and unaltered.
- Availability: A service must meet the requirement of stability and that information is available when the need arise.

It is very important for each organization to design a security policy that reflects the flow of information. This will aid the security managers and everyone affected by the configuration of the organization’s security. Firewalls, routing, software, hosts and many more features of information flow and storage will utilize this policy. And perhaps most important is the Intrusion Detection System itself. As the goal of the Intrusion Detection System is to be a burglar alarm it needs to know what’s defined as a threat, or attack. The off-the-self products, be it commercial or open source, have a basic setting defining misuse/anomaly, but to tune the system to act according to the organization’s best interests it needs to have a strict plan to set its parameters according to. A security Policy will enforce this transition and aid the technicians to configure a proper IDS.

4.1.4 Challenges

The term misuse/anomaly is relative to the network and the particular system that it will monitor. The IDS is not a wonder machine that only needs to be hooked up on the system. It needs configuration and regular updates. In many cases the manager and the IDS must be trained to discover the particular features and patterns in the network to be tuned well. These discovered features and patterns are then mapped into the IDS for optimal performance. But one must remember that generally a network is rarely static and the IDS must be updated continuously. The IDS has been developed from early days on for monitoring audits but as of later it has been used in several systems where security is an issue.

Today's Intrusions Detection Systems are powerful and capable of detecting most attacks, but they still need a lot of research. For both the anomaly and signature based Intrusion Detection Systems one must combat the high number of False Positives. The signature based IDSs have a flaw as they may let attacks more easily slip past unnoticed, but they tend to have a lower false positive than anomaly based, and they are faster. Most commercial Intrusion Detection Systems today are based on signatures [31]. The results are good, but the full potential of the IDS is yet to be fully unleashed. Several studies have been done on the algorithms for anomaly detection, but the answer to a perfect algorithm is far away, or may not exist. Still IDS face many challenges but the future looks good as it is still early on in the development/research process [12].

4.1.5 Elements of an IDS

An Intrusion Detection System consists primarily of three elements:

- A classification module that contains the algorithms that partition the data. This module may implement very different techniques such as signature/anomaly or clustering/learning.

- The analyzer interprets the results from the previous phase (classification) and labels the results accordingly. They might be labeled as normal or misuse/anomaly. Or they can have different degrees of labeling.

- The final element is the decision engine that makes a decision based on the security policy and the result from the analyzer.

A fourth element at the first stage might be added to optimize the process, see figure 1 below. The data filtering module has as a task to narrow down the specific features of the data so that the format closely resembles that used in the classification process [28]. Overhead data is removed and for example all relevant data is presented as numbers for further processing. This module might not be needed at all depending on the system used and the data it must analyze. The data source is the source from which the information is derived, for instance the network that the Intrusion Detection System will monitor.

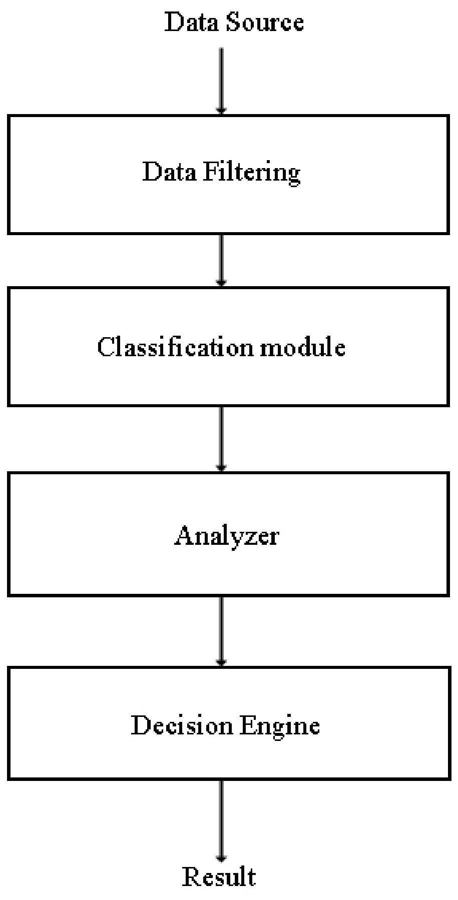


Figure 1: A basic overview of a standard Intrusion Detection System

There may be several sources in a particular system that gathers data. The objects that gather the data are often referred to as *sensors* and they may be computers that solely tap the network traffic or programs that log the events on computers. To tap the network traffic the network interface card is often set to *promiscuous mode* [3]. This makes it possible for the sensor to pick up all packages in the network, but as it does not contain an ip-address it makes it more difficult to attack. The drawback is that in a modern network where switches control the traffic, the sensor may not pick up any traffic, or only a limited portion of the overall network traffic.

The sensors, if placed in a large system, may be hierarchically set up, so that the lower level sensors send the information up to a sensor higher in the pyramid until the top sensor is reached. Then this sensor, placed at the top of the pyramid, has all the data gathered form the bottom sensors and the data are ready to be analyzed. The sensors is basically a sniffer or a simple log program.

The analyzer may also be hierarchically set up depending on the size of the entire system. If so the data are distributed among the analyzers and they may either produce a result of their own or run in parallel on the same data set in order to quicken the process. There are several ways to analyze data and this is the most difficult step. The main reason that analyzing is so difficult is that it is difficult to separate misuse, attack data, from normal data [3] [32]. They each have distinct features that separate them, but some features overlap, see figure 2.

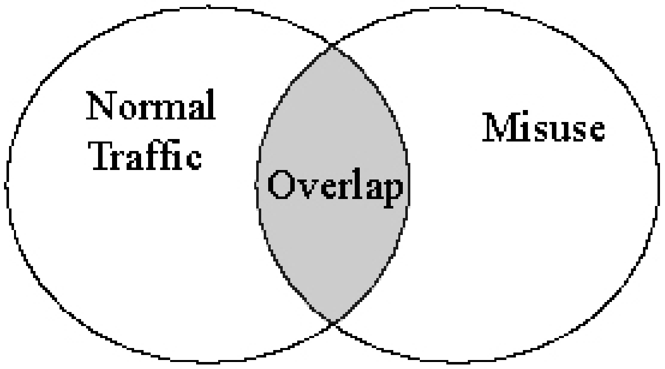


Figure 2: The problems separation misuse from normal traffic

What characterizes a misuse/anomaly may as well be a legal traffic form in a particular system. The analyzer must be customized for each of the networks it will supervise. This customization is derived from the network’s security policy and the particular network traffic.

It would seem that the future IDS will be of an anomaly detection based type, or perhaps a combination of signature and anomaly. Although anomaly has some issues like high false positive rate, it does have a potential that should not be overlooked. And especially with regard to the feature that it can detect new and yet unidentified attacks and abnormal traffic patterns. Clustering is a type of algorithm that can identify patterns and therefore is very lucrative for usage in an Intrusion Detection System. The clustering algorithms have the ability to discern misuse from normal traffic and classify data as attack or non-attack without prior knowledge. Clustering is valuable because its task is to categorize patterns unsupervised and this classification may be configured to discern attack from non-attack patterns in network traffic. The two clustering algorithms that this thesis concentrates on, k-means and Kernighan-Lin are two of the types that may be very good in an IDS. The k-means is used in several Intrusion Detection Systems today and with good results. Several researches and enhancements are done with the k-means algorithm as of today [6] [16]. What this thesis looks into is whether the Kernighan-Lin will provide a better result.

For the analyzing portion of an Intrusion Detection System much work still needs to be done. The market is still held back at signature based Detection which have good sense of detection of known attacks. The false positive rate for anomaly based Detection is still too high and this results in many alarms that are not an attack. This may yet again prove fatal for a system as people loose confidence in them. Although behind the numbers another potential threat emerges. A false negative rate is difficult to measure as this is actual attacks that bypass the detection. This number is assumed to be much higher in a signature based system, but it is difficult to prove. In a given normal network one cannot with a hundred percent certainty know what traffic is normal and what is abnormal. Therefore the number of attacks that are not detected cannot be determined accurately. One could use sanitized data/traffic but that is difficult to obtain and may be unfit for the network domain to be tested. The negative effect is that signature based is easier for a manager to relate to as it gives fewer false alarms and the attacks that are not detected will not bother the manager. But it gives a false sense of security.

The anomaly based IDS will alert often on non-attacks but it may often detect attacks that otherwise would pass unnoticed until it is too late. Clustering would seem to be the way to proceed with the anomaly based IDS and more emphasis on research is given in regard to this subject.

4.2 Clustering

Classification of data may be performed by two very different approaches [15]. It can be carried out with learning where the system slowly learns how the user wants to partition the features [17]. This system has some disadvantages when it comes to the initializing phase. It may be very time consuming to generate data the system will learn from. The manager of this system must know of every attack in the initial data set, and a large scale predefined dataset is difficult to come by. KDD-Cup is such a dataset. This dataset is based on a military traffic pattern and although it is good documented and skillfully categorized, the realistic use of this dataset is limited. Other networks, like commercial for instance, have a different traffic pattern and possibly very different attacks/misuses. The learning phase itself is also a time consuming task, as the system doesn't learn by itself. It needs a supervisor to decide right from wrong, or misuse from normal traffic. Systems based on learning have the potential to perform very well, but it is vulnerable as of how good the learning data set is, and how it is interpreted by the supervisor according to the security policy.

Another approach to classification of data is to use clustering techniques. Clustering is a set of procedures that partitions a set of data without prior knowledge [29] and is therefore an anomaly based IDS. Or it can be defined as an unsupervised classification of patters into groups [1] [14]. Clustering appeals to many researchers based on its versatility and because of the many fields it can be applied in [22]. Some areas include classification of properties of an image, document retrieval, classification of objects and persons such that recognition may be used, pattern reorganization/classification, data compression and data mining [1] [16] [29].

As data is provided for the clustering algorithm it will produce a set of classes. The distinction between a good clustering and a less good clustering algorithm is much based on the environment and the data that is provided. Some clustering algorithms may perform well in one setting that it is designed for, but severely lack performance when presented with another problem. No theoretical guidelines currently exist for choosing appropriate patterns and features. There is no clustering algorithm that is optimal for every task [19]. Therefore the algorithm in question must be closely examined with regards to the data and desired result before it is implemented. And more often that not the data must be cleansed before it is used as input in the clustering algorithm. Some data have to be altered and some have to be discarded/extracted.

The output of the clustering algorithm is closely tied to the input. It is then imperative for the user of the clustering algorithm to have a thorough understanding in both of the particular technique and detailed knowledge of the data gathering process. When the user has this desired knowledge it makes it easier to find the true class structure [1]. The most descriptive features that closely identify the data are the best to be utilized to increase the performance and validity of the class-structure. Features are then often arranged in a particular way before the clustering itself takes place [29]. Specific criteria are used as a check for the output's validity during a *cluster validity* analysis. One might often have a good idea of how some data are partitioned, but it is difficult to be sure of the entire outcome. To maintain the degree of objectivity in the validity assessment of an algorithm, or its output, one must look at the clustering structure. If this structure, or output, with reason can be said to not have occurred by merely a chance then it can be said to be valid.

The two figures below, figure 3:a and 3:b, describes a clustering process. Figure 3:a depicts X's spread out in a graph. These are points scattered about and with the use of clusters it is desired to make a categorization of these X's which can be concluded not to be random. Figure 3:b has been clustered and the numbers identify each of the clusters in which the X's belong. This is a constructed example but the idea of a clustering procedure shown. As one can see from figure 3:a a pattern is already present. And this pattern is formalized in figure 3:b, and it can be said that this categorization did not occur by mere chance.

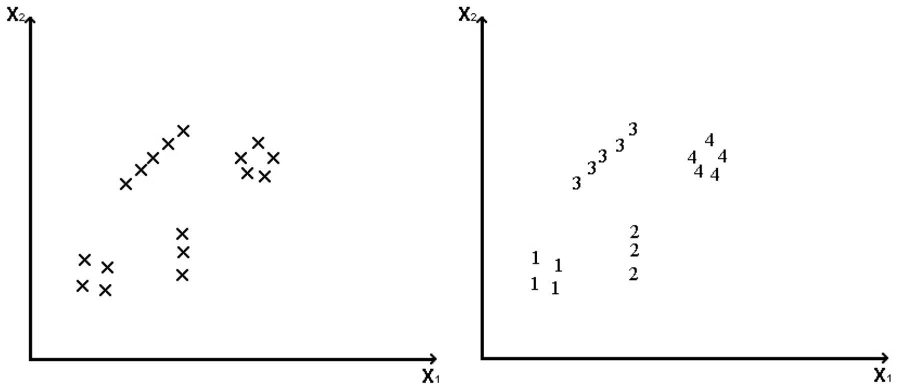


Figure 3 (a) and (b): Figure a, to the left, shows 17 features. Figure b shows 17 categorized features after clustering.

Clustering is mainly used to find patters in large data sets that may be overwhelmingly for a human to do. Patterns may be a measure of physical nature or an abstract notation. Patters are usually represented by multidimensional features, and each of these features represents a vector. In this study the vectors consists of 41 dimensions (features). The clustering algorithms are derived from assumptions and underlying proven mathematical fundamentals, but the algorithm as a whole is based on the best guess and analysis of results. The problem presented that a clustering algorithm must solve is often a time consuming task if the optimal solution is to be found. It is of an intractable nature as it cannot be done with a polynomial time algorithm on a deterministic computer [19]. The clustering algorithms are then based on finding several good partitions instead of one optimal. The algorithm may find the optimal solution but that is by chance [1]. The essential core of the problem is that the complexity of the problem is defined as non-polynomial or a high degree polynomial, $O(n^2)$ or more. This is too time-consuming to be practically useful.

The often used example *is the traveling salesman dilemma* to illustrate this: A salesperson must visit all towns but he doesn't know the optimal route. The towns are connected by various roads, some towns are connected with several other towns and some are just connected to one. To solve this problem an exhaustive search may be used and it is guaranteed to find the best available path for the salesman. But if the number of towns is large, it is nearly impossible to find all routes and then decide which one is the optimal. The complexity is polynomial and complexity quickly grows as numbers of towns are increased. Instead of testing all the possibilities a heuristic approach is proposed. That is trial and error, combined within the basic set of the algorithm.

The definition of a heuristic algorithm is given in [20]:

"An algorithm that usually, but not always, works or that gives nearly the right answer".

One might test several possibilities and see if one is better than the other. This testing may continue until desired results occur or other terminating criterion is met. Many clustering algorithms have a sort of random procedure or a heuristic approach, where different subsets are tested to find a solution.

This traveling salesman dilemma is just an illustration of the problem of why one might want to use heuristics to solve a problem. The clustering algorithms has a goal to solve a problem that is polynomial, within a short time frame as possible but still maintaining as a high degree of accuracy as possible. As previously mentioned, a heuristic method may not find the optimal solution, or partition, for the problem, but it will often come up with a good solution fairly quickly.

The clustering techniques will always suffer from tradeoffs. For example gain one increase of performance in accuracy, finding as close to the optimum result in nearly all cases, will often result in a severe decrease in speed [18].

There are several types of cluster algorithms and many variations of each one. This is natural since cluster algorithms must be customized based on the task at hand. Some different approaches to clustering algorithms are [1]:

F*uzzy* or *hard*; where hard clustering assign a pattern to one cluster, and fuzzy clustering assigns weights, or degree of membership, to clusters.

Deterministic or *stochastic*; when the clustering algorithm is deterministic it will follow a predefined set that should yield the same result when applied to the same data set. The stochastic algorithms use a random function to pick a starting subset for instance.

4.3 Distance

The clustering algorithm partitions the data based on similarities between the vectors from the same feature space. The distance measure must be chosen carefully in accordance to the problem at hand. Many clustering algorithms do not base the partition on similarities from the same feature space, but group them by dissimilarities(distances). For this distance calculation *Euclidean distance* is the most common. Euclidean distance measure is derived from the *Minkowski metric*. The Euclidean measure has its value p = 2 as shown with the formulas below [1]:

The Minkowski metric:

$$d_2(x_i, x_j) = \left(\sum_{K=1}^{d} | x_{i,k} - x_{j,k}|^p\right)^{1/p}$$

And the Euclidean measure where p = 2:

$$d_2(x_i, x_j) = \left(\sum_{K=1}^{d} | x_{i,k} - x_{j,k}|^2\right)^{1/2}$$

The Euclidean distance is easily understandable and therefore easily applied. It gives good measure and is computed efficiently. But it has some drawbacks as well. The Minkowski metric, and the Euclidean metric as its special case, may favor the largest-scale feature and cause the result to be less accurate. The values calculated should thus either normalize the features or use some weighting scheme to counter this [1].

4.4 Squared Error Algorithm

Squared Error Algorithm is a method that uses a criterion function in clustering techniques [1] [16]. The algorithm arranges the patterns with each predefined centroid. The centroid might be a selected vector from the dataset to be partitioned or a virtual vector of which the dataset must be arranged according to. The number of centroids is decided by the system, but normally an Intrusion Detection System would have two centroids; one that defines normal traffic and one that defines anomaly. The squared error criterion uses the formula shown below to find out which centroid is most similar to a given vector.

$$e^2(H, L) = \sum_{j=1}^{K} \sum_{i=1}^{nj} || x_i^{(j)} - c_j ||^2$$

L defines the initial clustering and **H** is the pattern, or data set to be categorized. **K** is the number of clusters. The centroid is **c** and **x** is the pattern.

4.5 The K-means Algorithm

Perhaps one of the most popular squared error algorithms for clustering is the k-means [1] [15] [16]. It is a very simple and intuitive algorithm. The implementation of the algorithm is easy and its time complexity is O(n), where n is the number of vectors to be clustered. These good properties make the k-means an attractive function within a system that utilizes clustering of data [16]. Because of it is linear time complexity, k-means can process great amounts of vectors without loss of performance. A computation of 1000 vectors would take double the time of computing 500 vectors.

The k-means starts with a random initial partition. The number of clusters is determined by the designer based upon the task of the algorithm. The task of k-means is to take the surrounding patterns and assign them to the initial clusters based on similarity. New centroids are then chosen and the algorithm re-run. This takes usually 3-5 iterations and then each centroids represents a cluster.

The weakness of k-means is the randomly chosen initial partition [1] [17] [26]. If that partition is chosen badly the result will be unsatisfactory. It will terminate with what it believes to be the minimum for the given dataset but that result is only a local minimum. The figure below, figure 4, shows potential local minima where k-means might halt and give a result. The global minimum is the correct answer.

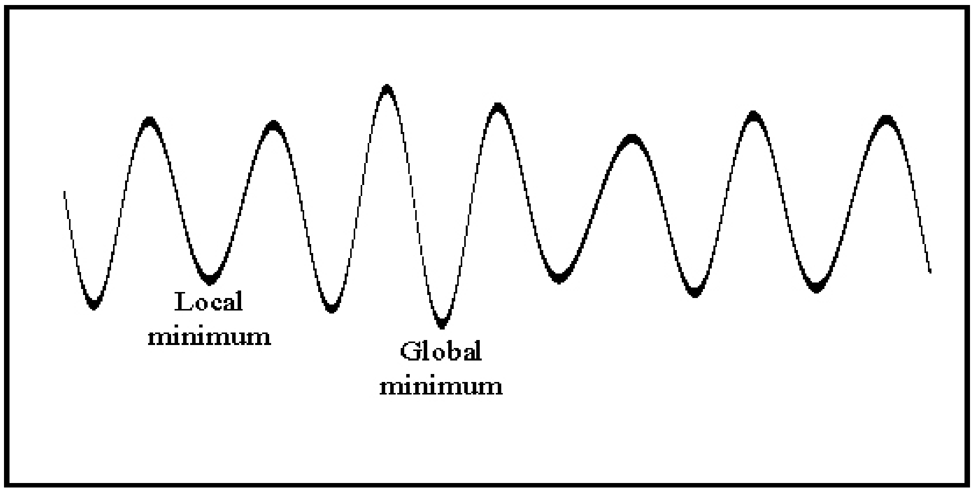


Figure 4: The k-means initialization problem

And often k-means will find this correct minimum, but if the random initialization is unfortunate then the result might be the spot market as a local minimum. This is a product of the heuristic approach and is valid for all heuristically based algorithms, but some algorithms are more affected by this weakness than others. This Thesis investigates if Kernighan-Lin is less affected by this local minimum problem and thus produces higher accuracy(less false positives).

A further illustration of the problem is shown in figure 5. If the initial selected patterns are A, B and C the result of the clustering performed by k-means will be as shown with the circles. K-means(A,B,C) = ({A}, {B,C}, {D, E, F}). But the global minimum or the optimal result for this situation is as shown with the rectangles.

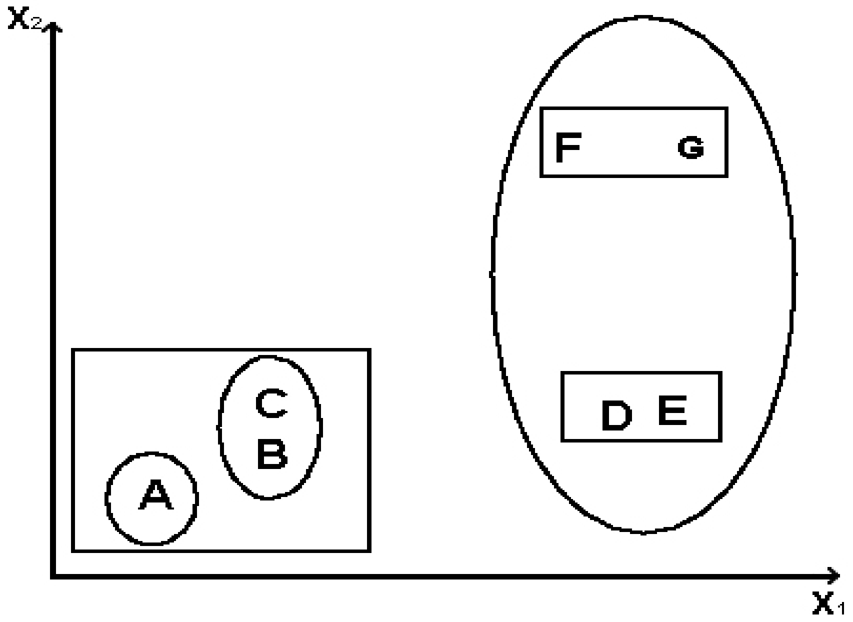


Figure 5: The k-means makes the wrong partition when the initial partition is badly chosen. The bad cluster result is within the circles and the optimal solution is within the rectangles

Squarederror(global minimum) = ({A, B, C}, {D, E}, {F,G}). If the k-means were to use A, D and F as initial partition it will yield the correct result.

K-means(A, D, F) = ({A, B, C} ,{D, E} ,{F,G}) [1].

4.5.1 The Kernighan-Lin Algorithm

The Kernighan-Lin algorithm is used to view external against internal cost between nodes of a graph that emerges from the given dataset to be clustered. The figure below, figure 6, shows two partitions; Partition A and Partition B. The nodes of the graph are assigned to each of the partitions. The internal cost is the cost of an edge between two nodes within the same partition. The external cost is the cost of the edge of a node in one partition to a node in the other partition. The Kernighan-Lin algorithm and used in this thesis uses two partitions only. The cost within each partition, the internal cost, is rated lower than the cost between nodes across the partition, the external cost.

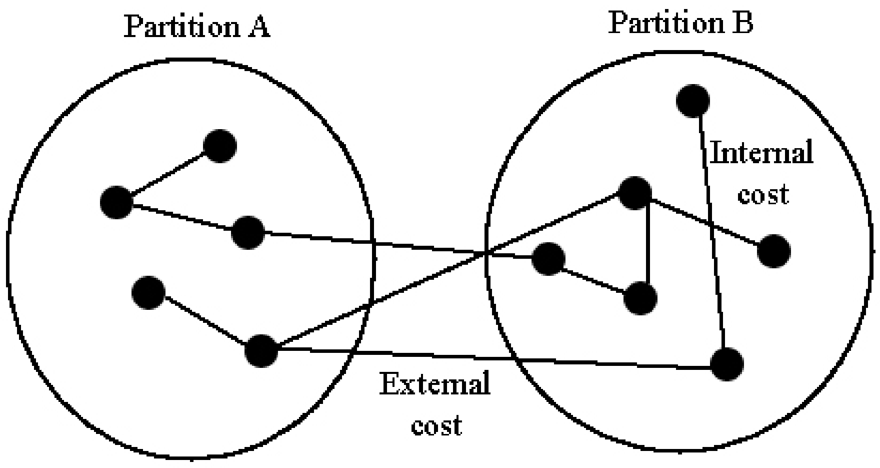


Figure 6: Illustration of External Cost and Internal Cost between two partitions

The algorithm tries to move each node between partitions so that the graph is maintained but with as low cost, the total sum of internal and external, as possible. Which in part mean that the internal cost should be maximized while the external cost is minimized. The Kernighan-Lin heuristic partitioning of interconnected graphs has been used in circuit partitioning for a time. And it has been proven to be very successful [7]. The results are close to the optimum and the time-complexity is almost linear.

The Kernighan-Lin can be easily implemented in hardware or software or a combination of the two, and over time the Kernighan-Lin has proven difficult to outperform. There are faster algorithms, and more accurate but as a combination Kernighan-Lin is one of the best. The results for Kernighan-Lin algorithm are a result of an extension provided by Fidduccia/Mattheyses and their addition made the Kernighan-Lin run in linear-time [7].

The unimproved Kernighan-Lin would give a time-complexity of $O(n^2)$, which is an polynomial time function and much slower than the k-means for instance. And the improved Kernighan-Lin is close to $O(n \log n)$. The implementation of the algorithm in this thesis follows the standard Kernighan-Lin algorithm. This specific version of the Kernighan-Lin algorithm is designed to produce the best possible result, but at the cost of time complexity. The flow-chart is given in figure 7 and its time complexity is very close to $O(n^2 \log n)$, which is by contrast of k-means a much slower process. This thesis also intends to use a high count of nodes n, so it is interesting to see if the difference in running time will be of concern.

4.5.2 Flow-Chart of Kernighan-Lin

The first Kernighan-Lin heuristic does is to partition the graph in to two parts. The size of the partitions is determined by the input. Then the pair of nodes with the highest external cost is found. This will then be one node from partition A and one from partition B, since the external cost is defined as the cost between nodes in separate partitions. The two selected nodes are subtracted from the initial partition group and the next node pairs are selected in the same manner. This continues until all nodes in one partition are exhausted and this requires one partition, partition A, to be smaller than the other. When all nodes in partition A are checked the algorithm further calculate the total of gains accumulated from this change and this values is **G**. If the value **G** is higher than 0 then further change can be made to improve the result. All the nodes pairs previously selected are exchanged and the algorithm runs again, increasing the iteration. If **G** is less than 0 then a local minimum are found and the algorithm are finished.

The flowchart is shown in figure 7 and a short description of symbols follows:

Let us consider a graph consisting of n nodes. Let **A** be the first partition of nodes and **B** the second one such that partitions **A** and **B** represent all nodes in the graph. Let **g** be the maximal gain for a single interchange between **A** and **B**. Let **D** be the difference of the external and internal cost between a node pair of partition **A** and **B** (**D** = External cost – Internal cost, for all nodes in the graph). The resulting **D** is then an array containing the cost difference for all nodes in the graph. Let **c** be the matrix for cost between every node. This is a two dimensional array and is often obtained by using the Euclidean distance described above. This cost can be a bidirectional cost that has a different value depending of which way one travels between two nodes.

The Kernighan-Lin heuristic will try do decrease the external cost by a series of interchanges of subsets of the partitions A and B. When no further improvement is possible with this interchange the resulting partitions is a local minimum with respect to the algorithm [2]. The resulting partitions will then have a fairly high probability of being a globally minimum partition.

The gain **g** is calculated by adding **D**, the difference of external and internal cost, of a single interchange between the partitions A and B. and then subtracts 2 times the cost matrix, **c**, for the subset. a_i and b_j are two nodes from the two partitions respectively that are changed, and those two nodes are selected such that **g** is maximum.

The reason for it to be added twice is that **D** is expressed as an internal cost in the subset and after the exchange it will be an external cost. The maximum gain for a single exchange between the partitions is then calculated by:

$$g_1 = D_{a_i} D_{b_j} - 2C_{a_i b_j}$$

D is then recalculated after the change for all nodes in both partitions by the following formula.

$$D_x = D_x + 2C_{xa_i} - 2C_{xb_j}$$
$$D_y = D_y + 2C_{yb_j} - 2C_{xa_i}$$

x is an element within the partition A subtracted the sub-partition a_i and Y is an element within the partition B subtracted the sub-partition b_j.

When all nodes have been exhaustedly calculated and we have a gain for all of the subset pairs within the partitions A and B respectively, the maximum gain from the set **g** is selected. The inner loop of the algorithms terminates when all the nodes in the smallest partition have been selected (p = n). The smallest partition has to be A for this to terminate. If this maximum gain is greater than 0 further reductions can be made. If the maximum gain is 0 a locally minimum is reached and the algorithm is finished.

Usually this implementation has between 2-6 iterations before termination criteria are met [7] regardless of the number of nodes. Many experiments with different types of cost matrices have been performed. Both with 1-0 cost, integer cost and the results have been similar with regards to number of iterations, and indications from earlier work shows that it should not perform so differently with decimal value as cost [2].

This version of the Kernighan-Lin works only with two partitions, but for a use in an Intrusion Detection System that is of no concern, as it is desired to discern anomalous from normal traffic. The partition B must be larger than the partition A. If the partition A is larger than B, then the partitions must be swapped, and the necessary result must be reversed to produce the correct result. This is set by the algorithm itself and cannot work in any other way. This may propose a problem in an IDS as one cannot assume that there are more normal traffic than attacks. However the solution to this problem would be to make an initial check and if there are more attacks the partitions are swapped and results of the algorithm's output are changes accordingly.

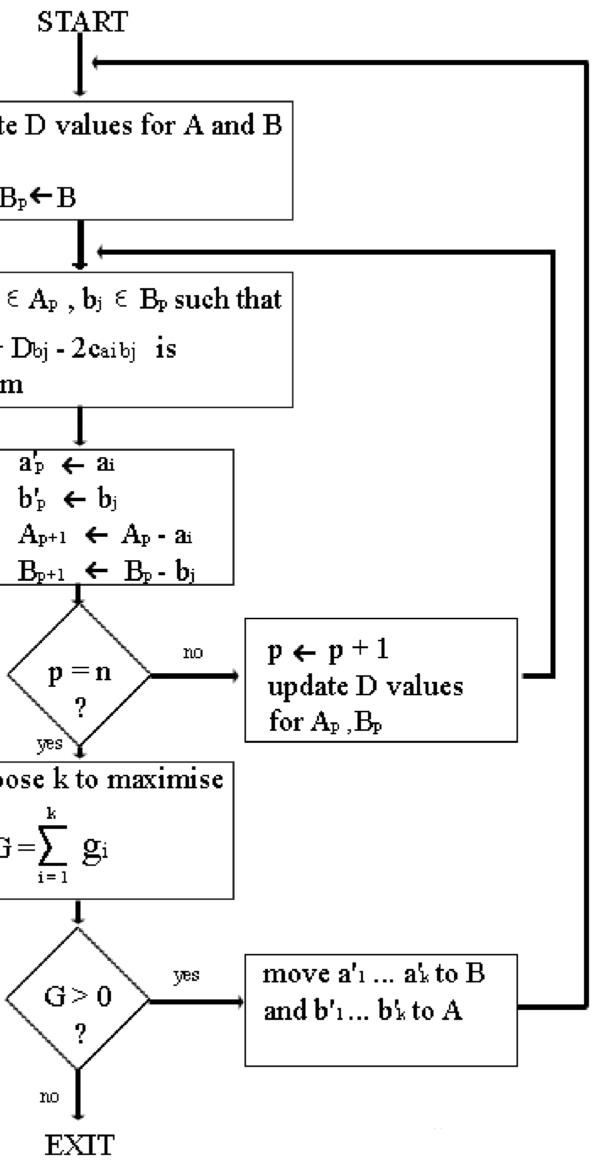

START
Compute D values for A and B
$p \leftarrow 1$
$A_p \leftarrow A$, $B_p \leftarrow B$
Select $a_i \in A_p$, $b_j \in B_p$ such that
$g_p = D_{a_i} + D_{b_j} - 2c_{a_i b_j}$ is maximum
$a'_p \leftarrow a_i$
$b'_p \leftarrow b_j$
$A_{p+1} \leftarrow A_p - a_i$
$B_{p+1} \leftarrow B_p - b_j$
$p = n$?
no
$p \leftarrow p + 1$
update D values for A_p, B_p
yes
Choose k to maximise
$G = \sum_{i=1}^{k} g_i$
$G > 0$?
yes
move $a'_1 \ldots a'_k$ to B and $b'_1 \ldots b'_k$ to A
no
EXIT

Figure 7: Flow-Chart of Kernighan-Lin Heuristic Partition Algorithm

5 Choice of Methods

5.1 Background

To answer the specified research questions one must establish an approach. This is necessary to maintain the required validity and reproducible experiments. The choice of experiment conducted is defined by Creswell in two principal methods [8]. The first approach is a *qualitative* one and is defied as one where the inquirer often makes use of knowledge claims based on multiple meanings of personal experience, or socially or historically constructed perspectives. This method is often used for development of a theory. Researched based on this method often ends up with open-ended, emerging data as a result. The *quantitative* method makes use of the cause and effect principle, where a hypothesis is generated and measurable variables are used to prove or disprove the claimed hypothesis. A *mixed-method* is a combination of both qualitative and quantitative where the researcher bases knowledge claims on more problem-centered and consequence-oriented grounds, collecting data either in the parallel or in a sequential order to get the best understanding of the problem, and collecting both numerical and text data.

5.2 Approach in Thesis

In this thesis there are chosen three questions to be answered, resulting in two hypotheses. Both of these hypotheses can be measured by the output of variables. By implementing the Kernighan-Lin heuristic partition algorithm in the Intrusion Detection System the output result could be measured. This thesis should then be able to answer whether the algorithm have a potential as a classification module. And further the results from Kernighan-Lin version of the IDS can be compared with the k-means version of the IDS. The outputs are measurable variables that needs no more translation and by interpreting the results this thesis gives an indication as of how good the Kernighan-Lin would be as a competitor against k-means. The variables are given as True Positive and False Positive, where true positive is the number of correctly labelled attacks and False Positive is the number of incorrect attacks. The IDS uses the classification module to determine the number of attacks and what of the subset, consisting of 1000 elements, are attacks.

The thesis relies mostly on a quantitative approach, as this gives the most accurate results. The variables to be measured and compared are already defined. The dataset to be used as basis for comparison between the two algorithms, Kernighan-Lin vs. k-means, is decided to be KDD-Cup 99. There are no other large dataset to use as a test set, and given the details of documentations this would be a good candidate.

Any written paper and documentations regarding clustering is very valuable for this thesis. This is a field in constant development and much interest is in this area. Many researchers have contributed in one way or the other and it is necessary to be updated on the available methods. It is crucial to get a deep insight of the workings of clusters, and clustering techniques, to fully understand the problem at hand. It is also desired to make the C-code run quick and avoid special fall pits. Using literature and research papers will be of great value in both assessing the problem and continue where no research-material is to be found, as well as answer those problems.

5.3 Coding

A study of clustering was necessary before any action were to be taken. The algorithm will be coded in, and the first prototype will be a simple version. This will be tested and if it passes the testing it will be further enhanced. This is to make sure that the procedure of Kernighan-Lin Is correctly done before any advanced calculations are done. When the more advanced version of Kernighan-Lin is finished it has to be tested within the IDS itself. Every function and calculation will be closely examined.

6 Experimental work

This chapter provides a description of the experiment carried out in this thesis, where the Kernighan-Lin algorithm is tested in order to see if it can be used in an Intrusion Detection System, and further how it compares with the opposing algorithm, k-means.

6.1 About this Thesis

This thesis performs an experiment to see which effect the Kernighan-Lin algorithm for partition of a graph would make in an IDS compared to the already implemented k-means. The k-means already gives good results but it still has some problems. As mentioned, the k-means has a weakness in how the initial partition is selected. The often used implementation makes a random initial selection in the graph and this selection may prove fatal. Also the k-means has a too high false positive rate. One might say that all results higher that 0 false positive is too high, but one must be realistic. The researchers and developers hope to reach this goal but on a practical level this will never be. As this may be it is important to keep this number as low as possible to make the system as reliable as possible. In this thesis we investigate if the Kernighan-Lin can truly provide better results or if some unknown issue makes it no more, or perhaps less, accurate than the k-means.

6.2 Purpose of the experiment

The purpose of this experiment is to evaluate the two hypotheses:

1. *Hypothesis: The Kernighan-Lin will provide a lower false positive rate than k-means.*

2. *Hypothesis: Kernighan-Lin will be slower than k-means.*

To do this, the Kernighan-Lin had to be coded with the goal to be used as a classification module in an Intrusion Detection System. The IDS was available as described in the previous chapter, but had the k-means as the classification module. What had to be done was to replace k-means with Kernighan-Lin and then use the test dataset for comparison. This Dataset is called the KDD-cup [6] [13] and is the only extensive dataset available. An Intrusion Detection System was obtained [4] for this thesis, and was modified to two versions, one with k-means and the other with Kernighan-Lin. The two versions were tested on the same computer and the results compared.

6.3 The coding

The Kernighan-Lin was coded in C-language using Microsoft Visual Studio 2005. The first version of coding was set up with no focus on the cost between the nodes, to test the capabilities at the lowest level. This program read from a file the nodes of the graph.

The file had a simple structure as shown in the table below. This table shows, in the first line, that node 1 and 3 are connected. The first version of the coded set the cost between these nodes as 1 and where there where no connection between nodes, the cost was set to 0.

File of graph
1 3
1 4
2 5
2 8
3 6
4 5
4 7
5 9
8 9

Table 1: The input file for the first version of Kernighan-Lin. 9 nodes and connections are read horizontally. 1 is connected to 3 and 4, 2 is connected to 5 and 8 and so on.

An old Fortran coded Kernighan-Lin program [5] was available that could be utilized and results of partitioning compared. In this way the C-coded version could be tested so that it would give the same results. The Kernighan-Lin was tested on a graph consisting of 9 nodes with connections as shown in table 1 and the partition sizes were set to 4 for A partition and 5 for the Partition B. The initial partition is shown in table 2(a). This is to partitions ordered 1-9 and the connection matrix was read from file, table 1. The result was the following partition shown in table 2(b):

Initial Partition:
Partition A: 1, 2, 3, 4
Partition B: 5, 6, 7 ,8 ,9

Table 2(a): The starting partition of the algorithm

Result Partition:
Partition A: 1, 7, 3, 6
Partition B: 5, 4, 2, 8, 9

Table 2(b): The resulting partition of the algorithm

This result was compared to the Fortran coded algorithm and they gave the same result. Both in the number of iterations (2), which is the number of exchanges between the partitions, and the resulting partition. Then, in order to have it work with an Intrusion Detection System, the C-coded Kernighan-Lin had to be modified. Firstly, and the most important was that the cost between nodes had to be calculated based on the 41 columns with attributes(features) for each node. This measure was calculated using the Euclidean distance as mentioned in Section 4.2. Some functions had to be revised for accurate calculation of the distance vector, and they were derived from the article [2]. The input function was changed from reading from a file to read from an array created by the IDS. Further the output was changed so that the resulting coding can be viewed in the appendix A, which contains the whole Kernighan-Lin algorithm. One important note regarding the code: The Kernighan-Lin algorithm requires that the first partition, partition A, is of a less size then partition B. The system was set up so that when the IDS called the Kernighan-Lin function it tested if A was smaller than B. If that was not the case it called another version of the Kernighan-Lin where the partitions were swapped and so was the output labeling for correct results.

6.4 KDD-Cup

KDD-Cup is the dataset used for this experiment [13]. The choice of dataset is greatly limited because of the difficulties and amount of work it takes to sanitize large amounts of data. The KDD-cup is a database containing almost 500.000 network packets which are cleansed and identified. The dataset contains a large set of attacks as it is derived from a military environment. This makes the dataset not good for an Intrusion Detection System based on learning. It would be suitable for an environment which is similar to the military but for other organizations this dataset would cause incomplete learning. The same criterion remains for testing a practical IDS for a given organization. This dataset may not represent the traffic pattern well. But as for research on unsupervised partitioning, cluster-based IDS, this dataset serves. The main goal of this thesis is to compare two algorithms, and the dataset in particular is of no great importance. What is important is the knowledge of number of attacks and what type they are when comparing two algorithms.

The vast amount of data and the detail of which it is described makes the KDD-Cup ideal for testing performance of Intrusion Detection System on a research basis. With almost half a million entries, each with 41 characteristics, makes a large test database and suitable for extensive tests of an Intrusion Detection System. The table shown below, table 3, gives an example of two network packets that are represented by KDD-Cup. The 43 rows represent an id for each packet, a classification if it is an attack or not and the 41 other rows make up the dimensions that makes a vector for a given packet in the Kernighan-Lin algorithm and the distance is calculated using Euclidean distance on all the columns for each packet.

id	19064	19219
Duration	1	22
protocol_type	0	0
Service	25	23
Flag	10	10
src_bytes	1405	140
dst_bytes	403	2020

Land	0	0
wrong_fragment	0	0
Urgent	0	0
Hot	0	0
num_failed_logins	0	0
logged_in	1	1
num_compromised	0	0
root_shell	0	0
Su_attempted	0	0
num_root	0	0
num_file_creations	0	0
num_shells	0	0
num_access_files	0	0
num_outbound_cmds	0	0
is_host_login	0	0
is_guest_login	0	0
Count	1	1
srv_count	1	1
serror_rate	0	0
srv_serror_rate	0	0
rerror_rate	0	0
srv_rerror_rate	0	0
same_srv_rate	100	100
diff_srv_rate	0	0

srv_diff_host_rate	0	0
dst_host_count	47	1
dst_host_srv_count	233	1
dst_host_same_srv_rate	100	100
dst_host_diff_srv_rate	0	0
dst_host_same_src_port_rate	0	100
dst_host_srv_diff_host_rate	0	0
dst_host_serror_rate	0	0
dst_host_srv_serror_rate	0	0
dst_host_rerror_rate	0	0
dst_host_srv_rerror_rate	0	0
Attack	12	12

Table 3: 2 examples of a KDD-CUP database entry

6.5 Initial

Both versions of the IDS were tested on the same computer, a standard IBM pc running Microsoft Windows XP. The application was coded within Microsoft Visual Studio 8. The test was not run in parallel but sequentially. The IDS with k-means implemented was tested first and then the results were noted. Then Kernighan-Lin version of the IDS was then tested and the results were compared. The dataset was the same for both versions, KDD-Cup.

The test was then redone on a different computer for a reference. This was also a computer running Microsoft Windows XP and Microsoft Visual Studio 8.

The IDS had some information as output given at the beginning and the end for each of the 1000 datasets. The important information here were the True positive and False Positive hits for the algorithm and number of attacks in the data set. In reality the packets in the environment that the Intrusion Detection System is guarding have four states when analyzed. Those four states are:

True negative: The IDS assumes that this is not an attack, and does nothing. This is correct since the traffic is not malicious.

False negative: This indicates that the IDS assume that traffic is not malicious, but in truth it is. This could for instance be an attack that goes by undetected. This is perhaps the most important state.

True positive: The traffic is labelled as attack, or a misuse by the IDS and this is correct. This is traffic of which the IDS should generate an alarm.

False Positive: This is normal traffic where an IDS generate an alarm as it assumes this is an attack/misuse. This is one aspect that one tries to reduce with an anomaly Intrusion Detection System.

The intersection of each of the states is given in figure 8 below. It shows the correlation between misuse and normal traffic, and that they are so closely tied that a distinct separation is very difficult. If one moves the decision threshold for an improvement in one instance, True Positive or True Negative, the effect will be that the opposite value will decrease in accuracy.

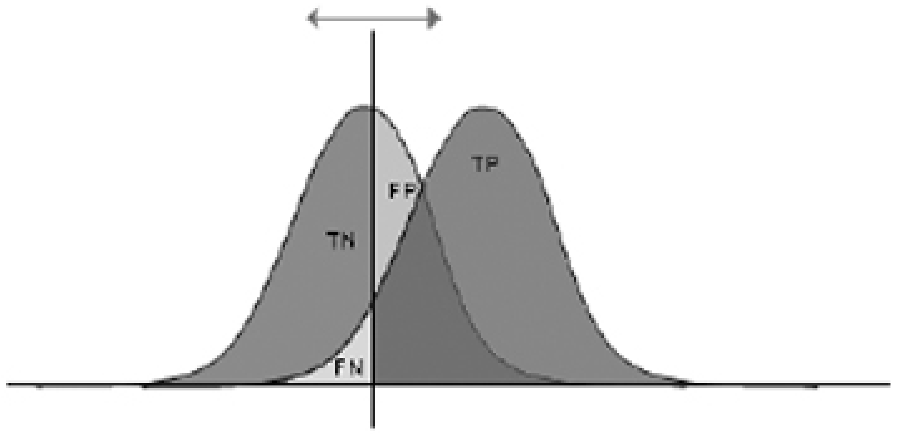


Figure 8: A graph showing the coherence between True Negative(TN), False Negative(FN), False Positive(FP) and True Positive(TP)

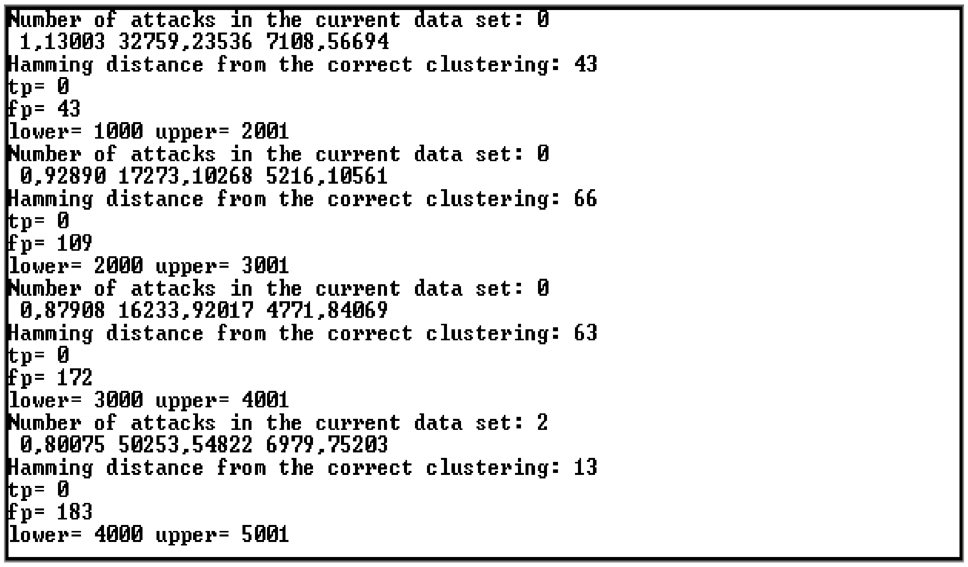
The IDS that is used in this thesis for testing the Kernighan-Lin Algorithm gives the output of True Positive and False Positive. The number of attacks are already known because of the well documented KDD-Cup test dataset. The True Positive and False Positive are related, a decrease in one leads to an increase on the other. The desired result is to have as low false positive as possible, hopefully 0, and a high true positive as possible.

6.6 Procedure

The Intrusion Detection System which has been coded prior to this thesis works by reading 1000 data packets from the dataset, KDD-Cup, and then analyzing this portion before another 1000 packets are read. In most papers [2] [7] the Kernighan-Lin heuristic was just tested with a much smaller dataset. The maximum was around 400 nodes. This test tries to see how Kernighan-Lin performes with as much as 1000 nodes in one partition. As Kernighan-Lin is a rather complex procedure, O(n^2 log n), it is interesting to see how it will perform with such a great number of nodes regarding the running time.

Every packet in the KDD-Cup dataset was run through by both versions of the IDS. Figure 9 displays the output for the k-means algorithm. The number of attacks ranged from 0-1000 in the data set. The parameters that are important in this figure is 'tp' (True Positives) and 'fp'(False Positives). 'tp' shows how many attacks the IDS have correctly identified and 'fp' shows how many attacks the IDS wrongly identify. This Intrusion Detection System compares the results from the algorithm with the test dataset. The algorithm has to find the correct number of attacks and label those exactly.

The last output sequence in this figure shows that there are 2 attacks in range 3000 – 4001 of the dataset. But k-means finds 11 attacks (183 – 172 = 11) but none of them are the actual attacks.



```
Number of attacks in the current data set: 0
 1,13003 32759,23536 7108,56694
Hamming distance from the correct clustering: 43
tp= 0
fp= 43
lower= 1000 upper= 2001
Number of attacks in the current data set: 0
 0,92890 17273,10268 5216,10561
Hamming distance from the correct clustering: 66
tp= 0
fp= 109
lower= 2000 upper= 3001
Number of attacks in the current data set: 0
 0,87908 16233,92017 4771,84069
Hamming distance from the correct clustering: 63
tp= 0
fp= 172
lower= 3000 upper= 4001
Number of attacks in the current data set: 2
 0,80075 50253,54822 6979,75203
Hamming distance from the correct clustering: 13
tp= 0
fp= 183
lower= 4000 upper= 5001
```

Figure 9: A portion of the output from the IDS running k-means algorithm

The difference between the two algorithms was mainly that Kernighan-Lin had number of attacks as input. Partition A had the size of the attacks and partition B had the size of non-attacks. Given 200 attacks in the dataset, partition A was 200 and B was 800. If the number of attacks was grater then 500 the partitions had to be reversed and the labeling reversed as well in order to accommodate this change.

Each run of the experiment had the same result both for Kernighan-Lin and k-means. The experiment was run 6 times each with the same results respectively. The IDS read a sequence of 1000 packages from KDD-Cup dataset for each run. The attacks within this sequence ranged from 0-1000, no attacks to all attacks.

6.7 Results of the Experiments

The results of the experiments show that Kernighan-Lin algorithm has a good potential as a classifier in an Intrusion Detection System. The speed was slower than the k-means but that difference was not too great by any standard. The partition on the other hand shows that it performs very well. When presented with the number of attacks it managed to correctly identify every attack.

Table 4 shows the 50 first data elements of k-means and Kernighan-Lin. Each element consists of 1000 packets. The results are incremental which means that for instance the True Positive in row two consists of True Positive from row 1 plus True Positive from row 2 and so on. This choice of output was already set in the IDS this study was based upon.

The TP(True Positive) is the number of attacks correctly identified by the algorithm and the FP(False Positive) is the number of falsely assumed attack. This means that if there are 5 attacks among the 1000 dataset and the IDS assumes 20 attacks it might get 5 TP and 15 FP if the labeling of the 5 attacks itself are correct. The ‘attacks’ column is the number of attacks in the dataset.

		K-means		Kernighan-Lin	
Data set	Attacks	TP	FP	TP	FP
0-999	0	0	43	0	0
1000-1999	0	0	109	0	0
2000-2999	0	0	172	0	0
3000-3999	2	0	183	2	0
4000-4999	0	0	201	2	0
5000-7999	0	0	308	2	0
6000-6999	0	0	316	2	0
7000-7999	376	374	316	378	0
8000-8999	1000	1376	316	1378	0
9000-9999	1000	2374	316	2378	0
10000-10999	1000	3374	316	3378	0
11000-11999	321	3695	329	3699	0
12000-12999	0	3695	364	3699	0
13000-13999	0	3695	428	3699	0
14000-14999	0	3695	432	3699	0
15000-15999	21	3695	494	3720	0
16000-16999	0	3695	565	3720	0
17000-17999	0	3695	644	3720	0

18000-18999	0	3695	688	3720	0
19000-19999	99	3794	702	3819	0
20000-20999	0	3794	745	3819	0
21000-21999	0	3794	807	3819	0
22000-22999	69	3810	822	3888	0
23000-23999	0	3810	903	3888	0
24000-24999	0	3810	947	3888	0
25000-25999	0	3810	948	3888	0
26000-26999	94	3878	986	3982	0
27000-27999	0	3878	1027	3982	0
28000-28999	0	3878	1050	3982	0
29000-29999	0	3878	1085	3982	0
30000-30999	0	3878	1108	3982	0
31000-31999	1	3878	1112	3983	0
32000-32999	0	3878	1205	3983	0
33000-33999	0	3878	1335	3983	0
34000-34999	0	3878	1384	3983	0
35000-35999	0	3878	1388	3983	0
36000-36999	0	3878	1509	3983	0
37000-37999	0	3878	1548	3983	0
38000-38999	0	3878	1565	3983	0

39000-39999	281	4154	1571	4263	0
40000-40999	724	4874	1571	4987	0
41000-41999	5	4874	1575	4992	0
42000-42999	127	5001	2448	5119	0
43000-43999	914	5915	2534	6033	0
44000-44999	1000	6915	2534	7033	0
45000-45999	1000	7915	2534	8033	0
46000-46999	1000	8915	2534	9033	0
47000-47999	1000	9887	2534	10033	0
48000-48999	1000	10887	2534	11033	0
49000-49999	1000	11887	2534	12033	0

Table 4: A comparison of the 50 first elements (50 000 first packets) of k-means and Kernighan-Lin

6.7.1 Accuracy

As noted the Kernighan-Lin performed very well. The experiment tested the whole dataset and when presented with the number of attacks it correctly identified all of them, resulting in 0 False Positives. The k-means had a much higher False Positive rate. Although the experiment is artificial as the Kernighan-Lin had the advantage to know the number of attacks it shows that the heuristic itself found all of the attacks. This is a great strength for an algorithm when used within an Intrusion Detection System.

From table 4 it is clear that Kernighan-Lin has a good accuracy as it detects all attacks. When presented with 376 attacks the Kernighan-Lin detects all but k-mean detects 374 of them. Both perform 100% accurate when presented with a dataset consisting of only attacks, 1000 attacks. But when there are 0 attacks Kernighan-Lin is the only algorithm which performs accurately. The k-means detects falsely 43 attacks as seen in the first row. By the three first rows the k-means has detected 172 attacks/ misuse which was in fact normal traffic. And with a low number of attacks, 2 in row 4, k-means doesn't detect any of the attacks but wrongly assumes 11 other attacks. The Kernighan-Lin identifies both attacks correctly.

The k-means heuristic performs well when the number of attacks is large. Then it mostly identifies all attacks and when the dataset have only contains attacks the k-means rarely assumes that the dataset include non-attacks. But at low attack rates within the dataset k-means suffers. Sometimes the False Positive rate is as high as 10%.

The advantage of Kernighan-Lin gained by knowing the exact number of attacks, while k-means did not, favors the Kernighan-Lin heuristic. The Kernighan-Lin had to know the partition size in advance and this could be accomplished by either using a fixed size, using k-means or the number of attacks itself. The choice was to use the actual number of attacks since this makes it possible to see of Kernighan-Lin can identify the attacks in the dataset correctly. It was not possible to do the same for k-means as it would require major changes in the coding of the algorithm. The testing would then be a compromise. The use of k-means as an indicator to the number of attacks and then use Kernighan-Lin to identify those would make the results too closely tied to the k-means.

6.7.2 Speed

The testing of speed of the two algorithms was of a secondary importance. This thesis would like to see if the Kernighan-Lin algorithm had to use significantly longer time to test the data set than k-means had to. The Kernighan-Lin had a much higher time complexity than k-means and given the size of the dataset it was assumed that the difference in running time between the two algorithms would be very noticeable. But as it turned out the delay in time was not that great. It was obvious that Kernighan-Lin had a slower procedure than the k-means but given that this version of the Kernighan-Lin algorithm is not optimized regarding time complexity the result here was very promising. The difference in time was close to around 30%-50% increase of running time when using Kernighan-Lin and this is far less than expected. Although this is not an accurate estimation of running time it gives an indication of the result. And even with 1000 nodes for each run indicates good results for Kernighan-Lin. As the running time of Kernighan-Lin increases with the number of nodes it must process, this number can be reduced for greater performance in time.

6.7.3 A combination of k-means and Kernighan-Lin

It was also tested to calculate the number of attacks using k-means and then let Kernighan-Lin separate attacks from non-attacks. The result was slightly better than k-means by itself. The number of attacks proposed by k-means is not accurate enough to provide good results. As mentioned k-means can be of by as much as 10-15%% and this again influence the results of Kernighan-Lin. But with the addition of Kernighan-Lin the attacks are identified more accurately.

7 Discussion of Results

The advantage for the Kernighan-Lin was that it was presented with the number of attacks, where k-means was not. The k-means did not allow an input of the exact number of attack and sometimes this would make it less accurate. But the remarkable discovery was that the Kernighan-Lin algorithm managed to identify with 100% accuracy every misuse from normal traffic. This means that it would certainly detect all the attacks if put in a loop and tried all the possible numbers of attacks, but at the cost of a reduced speed of course. In this case it would be 501 loops of the Kernighan-Lin heuristic as the dataset consists of 1000 elements. The first attempt would be A=0 and B=1000, next loop would be A=1 and B= 999 and so forth. Keep in mind that the partition A must be less than, or equal to, the partition B so the last partition is A=500 and B= 500. Another approach

A reduction of the number of elements in each sample would increase the speed of the Kernighan-Lin heuristic, but also increase the number of iterations of the dataset. The k-means was presented with a portion of data with 2 attacks and it found 11 attacks. But these 11 attacks were in fact no attacks at all and false positives were generated. The 2 attacks present would have gone unnoticed by and produced a true negative result. The attacks managed to slip through without being detected. These are especially the results of an Intrusion Detection System that one would like to avoid. Undetected attacks render a Detection System untrustworthy and even though it was only 2 packets that slipped through, this is not acceptable. Otherwise one could see from the table 4 that k-means produce a high amount of false positive. Which means that it tend to locate attacks where there are none. This has the effect to make managers of an Intrusion Detection system to loose confidence in the system. And it also takes a lot of resources to scan the alerts to find that there is no threat at all.

Kernighan-Lin on the other hand made no such mistakes, which is promising. Even if there were few attacks Kernighan-Lin algorithm detected them correctly. The loss of speed according to k-means is not as significant as first expected. The k-means was very fast as it scanned the dataset within seconds. The Kernighan-Lin algorithm used approximately same time on the given dataset. The time would be increased when Kernighan-Lin loops through all the possible partitions, and in this case the running time would increase about 500 times, some partitions in the loop would take very little time. A partition of A=1 and b=999 would often require less iterations than a a=400 and b=600 partition. But the Kernighan-Lin heuristic may also be configured to run faster regarding complexity and it may be other means than looping through all 501 combinations. Another algorithm may be used to detect number of attacks before Kernighan-Lin identifies them. K-means can be used in this way but this algorithm is too inaccurate for this task. It will not benefit the system too greatly as the accuracy improvement is minimal and the running time is much greater.

The Kernighan-Lin heuristic only works with two partitions, and this may be a disadvantage. In systems where the only goal is to separate misuse from normal traffic this cause no problem. But if the system requires several degrees of separation between different misuses Kernighan-Lin is unable to provide that functionality by itself. The solution here might be to categorize the results from Kernighan-Lin algorithm in a separate function. A proposed system would be that an algorithm first determines the number of attacks within the dataset, then Kernighan-Lin identifies the attacks and finally another algorithm categorizes the results in a desired classification. Depending on the complexity of the other algorithms this may be a slow process. Kernighan-Lin is not the fastest algorithm for classification and with the addition of two others the system may be greatly challenged regarding speed.

8 Conclusion

This thesis has demonstrated that it is possible to use Kernighan-Lin heuristic partition algorithm within an Intrusion Detection System and the results are very good regarding the accuracy. For this thesis 3 research questions were proposed in Section 1.4. Have the questions been answered and what, if that is the case, was the concluding answer?

1. *To what degree can Kernighan-Lin Algorithm be used in an Intrusion Detection System for detecting misuse?*

   The scenario set up for Kernighan-Lin algorithm to test if it could be used as a classification module in an Intrusion Detection System proved that it is possible. But we should keep in mind that the testing of this algorithm used the number of attacks as an input: the number of attacks is not known in a real environment. This version of Kernighan-Lin cannot work in an IDS by its own but the experiment clearly shows the power of the partitioning done with Kernighan-Lin heuristic. All the attacks were discovered and no false positives were generated. But Kernighan-Lin cannot by itself find the number of attacks. For this other means must be used.

2. *Analyze the k-means vs. Kernighan-Lin algorithms and see whether Kernighan-Lin has a higher or lower false positive rate than k-means.*

   The experiment conducted in this thesis proves that Kernighan-Lin have a much higher degree of accuracy than k-means. The Kernighan-Lin managed to identify all attacks and completely separated them from non-attacks. This shows that Kernighan-Lin is very good at discerning malicious traffic from normal traffic and the overlapping between the two was something that Kernighan-Lin algorithm managed to cope with. This was shown when there were massive attacks, no attacks and all in between. The Kernighan-Lin algorithm gives a higher true positive rate and a much lower false positive rate.

3. *Is there a significant speed difference between the two systems?*

   This was a secondary question that was asked to see if one would lose significant speed if accuracy was improved. The world of Intrusion Detection must struggle between several tradeoffs and one of them is the tradeoff between speed and accuracy [21]. Increase the performance in one causes loss of performance in the other. But this thesis showed that even though the accuracy was greatly improved the loss of speed was not that significant. This again proves the potential of Kernighan-Lin heuristic.

9 Future work

This thesis has shown that Kernighan-Lin algorithm may be a great solution for accuracy in an Intrusion Detection System. The increase in computation time is very small compared to the gain in accuracy. However the implementation of the Kernighan-Lin heuristic in this thesis is not final. It is proven that Kernighan-Lin heuristic can be used and the accuracy is high. But the speed may be further optimized. The coding itself is not optimized to run at greatest efficiency, and also it may have some security issues, for instance with creation of arrays using calloc-function. The implementation relies very much on the security measures in the main program feeding the information to the function. There are also several other variations of Kernighan-Lin algorithm that may be tested. These versions have focused more on the time complexity of the algorithm and sacrificed some of the accuracy. It remains to be tested if this lowered accuracy affects the result of the partitioning within an Intrusion Detection System. The faster variations of Kernighan-Lin algorithm have a time complexity that is very close to O(n).

As this implementation of Kernighan-Lin algorithm uses known attacks as an input it cannot be used directly as an IDS. What greatly separates the Kernighan-Lin from k-means is that it identifies attacks correctly. If a given dataset has 10 attacks and Kernighan-Lin algorithm is fed with knowledge of just 5 attacks it will correctly identify those 5 attacks. K-means algorithm may find 10 attacks but as shown with the experiment those 10 results could be non-attacks. A suggested further development of this research is to look at a way to enhance the implementation of Kernighan-Lin to work efficiently [7] [29] without prior knowledge of the dataset, i.e. in a loop. The use of k-means to calculate number of attacks is possible but it might be other ways to determine number of attacks in a more accurate way.

10 Bibliography

[1] A.K. Jain, M. N. Murty and P. J. Flynn. *Data Clustering: A review*. ACM Computing Surveys, Vol. 31, No. 3 sep 1999.

[2] B.W. Kernighan, S. Lin. *An Efficient Heuristic Procedure for Partitioning Graphs*. Bell System Technical Journal, vol. 49 No. 2 1970.

[3] R. G. Bace. *Intrusion Detection*. Macmillian Technical Publishing 2002.

[4] The Intrusion Detection System developed by S. Petrovic, G. Alvarez, J. Carbo, A. Orfila, February 2004.

[5] Fortran Code Developed by S. Petrovic.

[6] S. Petrovic, G. Alvarez, A. Orfila, J. Carbo. *Labelling Clusters in an Intrusion Detection System Using a Combination of Clustering Evaluation Techniques*. In Conference Proceedings of the 39th Hawaii International Conference on System Sciences 2006.

[7] F. Vahid, T. D. Le. Extending the Kernighan/Lin Heuristic for Hardware and Software Funtunal Partitioning. Kluwer Academic Publisers, Boston 1997.

[8] J. Creswell. Research design, quantitative, qualitative and mixed approaches. Sage publication, 2003.

[9] Description of the KDD-CUP dataset and the contest. http://www.kdnuggets.com/datasets/kddcup.html

[10] Bruce Schneier. *Secret and Lies*. Wiley Publishing Inc. 2000.

[11] T. Daler, R. Guldbrandsen, T. A. Høie, B. Melgård, T. Sjølstad. *Håndbok i datasikkerhet*. Tapir akademiske forlag 2002.

[12] S. Northcut, L. Zeltser, S. Winters, K.K Frederick, R.W. Ritchey. *Inside Network Perimeter Security*. New Riders Publishing 2003.

[13] Download of the KDD-Cup dataset. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[14] A. Strehl. Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining. 2002. http://www.lans.ece.utexas.edu/~strehl/pubs.html

[15] P. Berkhin. *Survey of Clustering Data Mining Techniques*. Accrue Software Inc 2002. http://www.ee.ucr.edu/~barth/EE242/clustering_survey.pdf

[16] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A, Y. Wu. *An Efficient k-Means Clustering Algorithm: Analysis and Implementation*. IEEE vol. 24, no. 7. July 2002.

[17] G. Fung. *A Comprehensive Overview of Basic Clustering Algorithms*. June 22, 2001.
http://www.cs.wisc.edu/~gfung/clustering.pdf

[18] R. Xu, D. Wunsch II. *Survey of Clustering Algorithms*. IEEE, vol. 16, No. 3, May 2005

[19] V. Estivill-Castro. *Why so many clustering algorithms*. ACM SIGKDD Exploration Newletter vol. 4, No. 1, June 2002.

[20] NIST. Definition of heuristic.
http://www.nist.gov/dads/HTML/heuristic.html

[21] H. Debar, M. Dacier, A. Wespi, S. Lampart. *An Experimentation Workbench for Intrusion Detection Systems*. IBM Research Report March 1998.

[22] Y. Guan, Large-Scale Clustering: Algorithm and Applications. The University of Texas, May 2006.

[23] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, M. A. Zissman. *Evaluation Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation*. Computer Networks 34 (4), 1999. 579U595.

[24] H. S. Javitz, A. Valdes. The SRI IDES Statistical Anomaly Detector. IEEE 1991

[25] J. McHugh, A. Christie, J. Allen. *The Role of Intrusion Detection Systems*. Software Engineering Institute CERT, IEEE October 2000.

[26] A. K. Jain, a Topchy, M. H. C. Law, J. M. Buhmann. *Landscape of Clustering Algorithms*. Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR Aug. 2004.

[27] G. Jagannathan, K. Pilliaipallamnatt, R. N. Wright. *A New Privacy-Preserving Distributed k-Clustering Algorithm*. SIAM SDM Conference 2006. http://www.cs.stevens.edu/~rwright/Publications/sdm06.pdf

[28] R. Vaarandi. *A Data Clustering Algorithm for Mining Patterns From Event Logs*. IEEE 2003.
http://kodu.neti.ee/~risto/publications/slct-ipom03-web.pdf

[29] S. Oliveira, S. C. Seok. *Matrix-based algorithm for document clustering*. March 2005.
http://www.cerfacs.fr/algor/CSC05/Abstracts/27_oliveira.pdf

[30] S. Bandyopadhyay, E. J. Coyle. *An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks*. Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003).

[31] S. Antonatos, K. G. Anagnostakis, E. P. Markatos. *Generating Realistic Workloads for Network Intrusion Detection Systems*. AMC workshop on Software and Performance, 2004.
citeseer.ist.psu.edu/antonatos04generating.html

[32] D. E. Denning. *An Intrusion Detection Model*. IEEE Transaction of Software Engineering Se-13, 1987

APPENDIX

A. CODE

```c
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>

/* Kernighan-Lin algorithm with cost differentiated nodes. Includes calloc array allocation. This
Function must be called differently depending on the size of na_r. If na_r is equal or larger than
nb_r then the partitions must be swapped and labeling at the end must be swapped as a result.
na_r is the first partition and nb_r is the second partition. The double pointer indata contains the
twodimentional array of the data to be partitioned. Maxlength is the length of colums indata, and ndata is
the number of rows, or essentially the number of nodes. Partdef is the output label in which the result
partition is labeled. Its labeled 1 for attack and 2 for non attack. */

void kl(int na_r, int nb_r, double **indata, int ndata,int *partdef, int maxlength){
	int na = na_r;
	int nb = nb_r;
	float max , gsum;
	int a = 0;
	int b = 0;
	int i , k, g;
	int l = 0;
	float **c;
	int *apart;
	int *apart1;
	int *bpart;
	int *bpart1;
	int *xset;
	int *yset;
	float *Ddata;
	float *gains;
```

```
c = (float **)calloc(ndata,sizeof(float *)) ; //2d array for cost matrix
      for(g=0;g<ndata;g++)
            c[g] = (float *)calloc(ndata ,sizeof(float));

apart = (int *) calloc(na +1, sizeof(int)); //allocate memry for arrays
apart1 = (int *) calloc(na +1, sizeof(int));
bpart = (int *) calloc(nb+1, sizeof(int));
bpart1 = (int *) calloc(nb+1, sizeof(int));
xset = (int *) calloc(ndata, sizeof(int));

yset = (int *) calloc(ndata, sizeof(int));
Ddata = (float *) calloc(ndata, sizeof(float));
gains = (float *) calloc(ndata, sizeof(float));

setvar(na, nb, ndata, maxlength, apart, apart1, bpart, bpart1, c, xset, yset);
graphread(ndata, maxlength, c, indata);
initial(ndata, na, nb, apart, apart1, bpart, bpart1);

do   {
     i = 0;
     //calculates D = external cost - Internal Cost for every point of the graph
     dvalues(ndata, na, nb, Ddata, apart, bpart, c);
     do      {
             //Find a maximal gain as a result of possible exchange of the ponts
             //from Apart and Bpart
             max = maxgain(&a, &b, na, nb, Ddata, apart, bpart, c);
             gains[i] = max;
             xset[i] = a;
             yset[i] = b;
             //remove a from Apart and b from Bpart
             setdif(apart, a, na);
             setdif(bpart, b, nb);
             calc(na, nb, a, b, apart1, bpart1, c, Ddata);
             i++;
             if(i > (ndata-1)) //if array out of bounds then exit
                   exit(0);
             }while(!isempty(apart, na));
```

```
        //Find the maximal partial sum in the array gains
        maxpartsum(ndata, na, &gsum, &k, gains);
        ++l;
        //if Gsum > 0 exchange and repeat.
        if(gsum > 0){
                     exchange(na, k, apart, apart1, xset, yset);
                     exchange(nb, k, bpart, bpart1, yset, xset);
                     }
       }while(gsum > 0);


    for(g = 0; g < na; g++) partdef[apart1[g]] = 1; //label the output, 1 is attack and 2 is non-attack
    for(g = 0; g < nb; g++) partdef[bpart1[g]] = 2;

    free(apart); //free the memory
    free(apart1);
    free(bpart);
    free(bpart1);
    free(xset);
    free(yset);
    free(Ddata);
    free(gains);
    for(i=0;i<ndata;i++)
           free(c[i]) ;
    free(c) ;
    } // end kl
```

```c
void graphread(int ndata, int maxlength, float **c, double **indata){
    /*structurize the input from the sourcefile. The data is from indata. maxlength is the number of colums
    and ndata is number of rows. The distance calculation uses euclidian measure*/
    int i, j, k;
    float sum;
    double s;

    for(i = 1; i < ndata; i++){
          for(j = (i + 1); j <= ndata; j++){
                s = 0;
                for( k = 0; k < maxlength; k++) {
                      s += pow((indata[i][k] - indata[j][k]) ,2);
                      }//k
                if(s >= 0) sum = (float)sqrt(s);
                else {
                      printf("Unable to square root: %f", s);
                      exit(0);
                      }
                c[i-1][j-1] = sum;
                c[j-1][i-1] = sum; //the array has same cost for both direction.
          }//j
    }//i
}//end graphread
```

```
void initial(int ndata, int na, int nb, int *apart, int *apart1, int *bpart, int *bpart1){
    /*Forms the initial structure of the graph. */

    int i = 0;
    int j = 0;
    int k;

    for(k = 1; k <= ndata; k++) {
          if(k <= na){
                 apart[i] = k;
                 apart1[i] = k;
                 i++;
                 }
          else {
                 bpart[j] = k;
                 bpart1[j] = k;
                 j++;
                 }
          }
     }//end initial

void dvalues(int ndata, int na, int nb, float *Ddata, int *apart, int *bpart, float **c) {
    /*Calculates the value D for every point of the graph by using
    the Internal and External values. Every node have edges to one another*/

    int i;
    int x;
    int y;
    float *Ea = (float *) calloc(ndata, sizeof(float));
    float *Eb = (float *) calloc(ndata, sizeof(float));
    float *Ia = (float *) calloc(ndata, sizeof(float));
    float *Ib = (float *) calloc(ndata, sizeof(float));
```

```
for(i = 0; i < ndata; i++) {
               Ea[i] = 0;
               Eb[i] = 0;
               Ia[i] = 0;
               Ib[i] = 0;
               Ddata[i] = 0;
               }

for(x = 1; x < (ndata -1); x++) {
      for(y = (x+1); y < ndata; y++) {
            if(ismember(x, apart, na) && ismember(y, bpart, nb)) {
                            Ea[x - 1] += c[x - 1][y-1];
                            Eb[y - 1] += c[y - 1][x-1];
                            }
            if(ismember(y, apart, na) && ismember(x, bpart, nb)) {
                            Ea[y - 1] += c[y - 1][x-1];
                            Eb[x - 1] += c[x - 1][y-1];
                            }
            if(ismember(x, apart, na) && ismember(y, apart, na)) {
                            Ia[x - 1] += c[x - 1][y-1];
                            Ia[y - 1] += c[y - 1][x-1];
                            }
            if(ismember(x, bpart, nb) && ismember(y, bpart, nb)) {
                            Ib[x - 1] += c[x - 1][y-1];
                            Ib[y - 1] += c[y - 1][x-1];
                            }
            }
}
for(i = 1; i <= ndata; i++) {
      if(ismember(i, apart, na)){
            Ddata[i - 1] = Ea[i - 1] - Ia[i - 1];
            }
      else {
            Ddata[i - 1] = Eb[i - 1] - Ib[i - 1];
            }
      }
```

```
    free(Ea);
    free(Eb);
    free(Ia);
    free(Ib);
}// end dvalues

float maxgain(int *a, int *b, int na, int nb, float *Ddata, int *apart, int *bpart, float **c) {
    /*Calculates the maximal gain obtained after exchange of element a from Apart and b from Bpart*/

    float gain;
    float max = -999999999.0;
    int i;
    int j;

    for(i = 0; i < na; i++) {
          for(j = 0; j < nb; j++) {
             if(apart[i] != 0 && bpart[j] != 0){
                gain = Ddata[apart[i] - 1] + Ddata[bpart[j] - 1] - (2 * (c[apart[i] - 1][bpart[j] - 1]));

             if(gain >= max){
                      max = gain;
                     *a = apart[i];
                     *b = bpart[j];
                     }
             }
          }
    }
    return max;
}//end maxgain
```

```
void maxpartsum(int ndata, int na, float *gsum, int *k, float *gains) {
    /*Calculates the maximal partial sum of the mambers of
    the array gains*/

    float s;
    int i;
    int j;
    float *sum = (float *) calloc(ndata, sizeof(float));
    *gsum = -9999;

    for(i = 0; i < na; i++) {
          s = 0;
          for(j = 0; j <= i; j++) {
                s = s + gains[j];
               }
           sum[i] = s;
           }
    for(i = 0; i < na; i++) {
          if(sum[i] >= *gsum) {
                *gsum = sum[i];
                *k = i;
                }
          }
    free(sum);
}//end maxpartsum
```

```
void exchange(int nab, int k, int *part, int *part1, int *set, int *set1) {
     /*Exchange the sets xset and yset between Apart and Bpart*/

     int i;
     int j;
     for(i = 0; i <= k; i++) {
           for(j = 0; j < nab; j++) {
                 if(set[i] == part1[j]) {
                       part[j] = set1[i];
                       part1[j] = set1[i];
                       }
                 else part[j] = part1[j];
                 }
           }
     for(i = 0; i < nab; i++) part1[i] = part[i];
}//end exchange


void calc(int na, int nb, int a, int b, int *apart1, int *bpart1, float **c, float *Ddata){
     /*Recalculates the d values after the maximal gain is chosen*/

     int i;

     for(i = 0; i < na; i++) {
          if(apart1[i] != a)
            Ddata[apart1[i]-1] = Ddata[apart1[i]-1] + (2*c[apart1[i]-1][a-1]) - (2*c[apart1[i]-1][b-1]);
     }

     for(i = 0; i < nb; i++) {
          if(bpart1[i] != b)
            Ddata[bpart1[i]-1] = Ddata[bpart1[i]-1] + (2*c[bpart1[i]-1][b-1]) - (2*c[bpart1[i]-1][a-1]);
     }

}//end calc
```

```c
void setdif(int *s, int x, int n) {
    /*removes the element x from the set , but number of elements is the same*/

    int i;

    for(i = 0; i < n; i++) {
          if(x == s[i]) {
                s[i] = 0;
                break;
                }
          }
}//end setdif

int isempty(int *s , int n) {
    /*returns 0(false) if array s is not emtpy. otherwise it returns 1(true)
    Checks whether set s is empty*/

    int i;

    for(i = 0; i < n; i++) {
          if(s[i] != 0)
                return 0; //false
                }
          return 1;    //true
}// end isempty

int ismember(int x, int *s, int n) {
    /*Determines if x is an element in the set s. If x is an
    element the function returns 1, true. Otherwise it returns
    0, false*/

    int i;
    for(i = 0; i < n; i++ ) {
          if(x == s[i]) return 1; //true
          }
    return 0;   //false
}// end ismember
```

```c
void setvar(int na, int nb, int ndata, int maxlength, int *apart,int *apart1, int *bpart, int *bpart1,
    float **c, int *xset, int *yset) {
    /* Make certain that all elements are at 0 when initilized*/

    int i;
    int j;
    for(i = 0; i < na; i++){
          apart[i] = 0;
          apart1[i] = 0;
          }
          for(i = 0; i < nb; i++){
          bpart[i] = 0;
          bpart1[i] = 0;
          }

    for(i = 0; i < ndata; i++){
          xset[i] = 0;
          yset[i] = 0;
          for(j = 0; j < maxlength; j++) {
                c[i][j] = 0;
                }
          }
}//end setvar
```