Master's Thesis

Applying Clustering Techniques for Refining Large Data Set (Case Study on Malware)

1710443 Yoon Myet Thwe

Supervisor	Mizuhito Ogawa
Main Examiner	Mizuhito Ogawa
Examiners	Nao Hirokawa
	Nguyen Minh Le
	Kazuhiro Ogata

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

August 2019

Abstract

Since the last decades, malware has been growing exponentially with fast infection rate. With the successive evolution, the modern malware created with obfuscation techniques becomes a great challenge for antivirus software vendors and malware researchers. Malware analysts seek for the malware samples to inspect the malicious behaviors and threat techniques and try to develop defenses against malware attacks. For the same purpose, many malware databases are collecting the new malware samples periodically and share the samples with the malware analysts to aid in their research. Intentionally, these malware sources have been collecting and storing the garbage files together with the malware samples. Unexpected interruptions like network failures can cut out the downloading process of malware sample, resulting as an incomplete file as a prefix of the other. We call it a garbage, which would lead to incorrect bias when applying statistical analyses.

Our goal of this research is to refine the malware data sets by finding the garbage files among the new collected samples. Each garbage is a prefix of a complete malware and checking a pair is an easy binary pattern matching. However, if target data sets become huge, the number of combinations to compare grows in the square manner. Instead, we investigate an application of clustering techniques as a preprocessing. Then, each data set is decomposed into certain numbers of clusters consisting of similar binary codes, and the binary pattern matching of pairs of malware is limited to each cluster. Our target data sets start with raw data sets of IoT malware (5,763 malware samples) collected at Yokohama National University and further check on data taken from VirusShare, from 2012 June 15 till November 27 and from 2019 January 20 to February 12, as the total more than 1TB. They consist of 30 folders containing either 65536 or 131072 samples in each. In addition, we combined two to five of these data folders to see the behavior.

We observed that the data sets from Virusshare include 1 to 8 percent amount of garbage while IoT malware raw data set provided by Yokohama National University has more than 40 percent of garbage. These experiments drive us to speed up the binary pattern matching algorithm to be able to handle larger data sets.

We tested five unsupervised clustering methods: k-means, hierarchical clustering, DBSCAN, spectral clustering and Birch. Among these methods, k-means turns out to be the most suitable algorithm to cluster the malware data sets in terms of runtime and accuracy. Although it can separate the

data set more accurately than other algorithms, some resulting clusters are unbalanced with the number of files in them.

We implemented nested clustering algorithm to reduce the unbalanced data clusters. We applied k-means iteratively until the number of files in every resulting cluster is less than the limited amount. Then, the binary pattern matching is processed to find the garbage in each cluster. Based on our experimental results, it can be seen that the combination of nested clustering and matching process executes two to three times faster than the matching process by oneself. Besides, it maintains the higher accuracy of matching garbage.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Mizuhito Ogawa for the continuous support of my research, for his patience, and immense knowledge. He helped me come up with the thesis topic and guided me in all the time of research and writing of this thesis. I am deeply grateful for all of his cares on me during my staying at JAIST.

Besides my advisor, I would like to thank Associate ProfessorNao Hiroka, and Associate Professor Nguyen Le Minh, for their precious suggestions and insightful comments, which give me a great help in improving my work.

I would like to show my appreciation to my labmates, Mr. Trac Quang Thinh, Mr. NguyenLam Hoang Yen, Mr. Vu Viet Anh and former intern, Mr. Pham Ngoc Dung for stimulating discussions, and all the support during the time I am staying at JAIST.

Finally, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Malware, also known as malicious software represents one of the most harmful programs that threaten the individuals' privacy and computer's security. Surfing the internet, the number of new malicious software has increased exponentially making cybersecurity a target for spreading these threats. Malware infections are among the most frequently encountered threats in computer security and it has also been increasing periodically. The increased number of malware samples have been creating many challenges for antivirus companies.

CNN Tech stated that more than 317 million of new malware was detected last year implying that a huge amount of malware were generated day by day[1]. On the other hand, many malware collecting systems like honeypot have been collecting the newly created large amount of malicious programs (malware) for the purpose of registering them after examining and classifying the malware according to their characteristics. Developing new techniques and defenses against malware threats will be followed up afterward.

While collecting malware files through the network, some of them might not be completely transferred to the destination due to some interruptions such as a network outage or system failures. As a result, the prefix of some malware will be collected as a garbage file together with the next attempt of collecting the complete malware. The proportions of collected garbage files are not small enough to be negligible. Finding such garbage files from malware databases as a preprocessing step could bring accurate analytical results and high efficiency for malware analyses.

This research aimed to implement the method for refining the data set by detecting garbage among the mixture of malware with garbage files. We

proposed a novel method for detecting garbage from large malware databases using malware binary information.

1.2 Contributions

Dealing with large databases required speedy and scalable techniques. This study focuses on modifying the garbage detection technique to be fast enough to handle huge data set. Our contributions are summarized as follows:

- Grouping the data set into relevant small clusters using nested k-means clustering algorithm
- Combing binary pattern matching together with quick sort algorithm to detect garbage from each cluster simultaneously
- Comparing the efficiency of nested k-means clustering algorithm other clustering algorithms

As a final step, we evaluated the performance and accuracy of our proposed method for refining the malware databases.

Chapter 2

Malware Database

2.1 PC and IoT malware

Over the last decades, computers have become increasingly popular and computer technology has made several important impacts on our society. Meanwhile, the Internet has been established as an indispensable part of daily life. Not long after the introduction of the PC, computer viruses evolved and have become ever more sophisticated and troublesome. Cybercriminals designed computer programs, called malicious software or malware, to penetrate and harm computers without user's content. So far, malicious software is mostly focused to target PCs running on Microsoft Windows OS, the most widely used operating system.

As the Internet becomes an indispensable tool, the focus of malware authors and operators slowly but steadily started expanding towards the Internet of Things (IoT) malware. The Internet of Things (IoT) is a system of extending the interconnection among computing devices through the Internet to share the resources and improve user experiences. IoT devices are everywhere and are taking part in more and more aspects of modern life every single day. As the current IoT devices are typically CPU-controlled micro-computers, many existing malicious threats infecting the computers through the Internet can also attack the IoT devices[2]. Malware developers take advantage of IoT and advanced interconnecting protocol to widely spread their malicious software.

2.2 Evolution of the Threats

As technology advanced with years, malware becomes more complex. Malware authors use sophisticated techniques to enhance the malware effects,

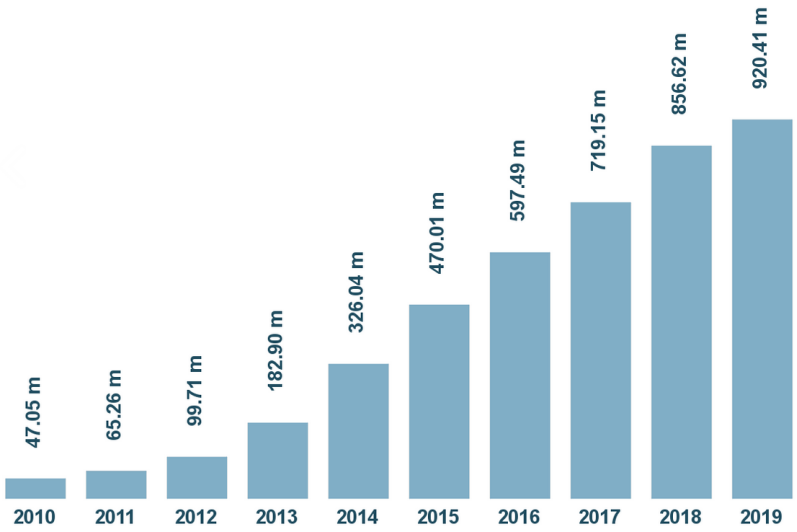

Figure 2.1: Malware evolution throughout 10 years

with increasing variety. According to the Cisco 2018 annual cybersecurity report[3], the evolution of malware was one of the most significant developments. The abundant amount of malware grew significantly during 2000 and 2010. From then and till now, the sophisticated malware has been evolving rapidly[4].

AV-test institute collected and analyzed the evolution of malware throughout ten years. Form the (Figure 2.1), we can see the excessive growing number of malware year by year.

2.3 Garbage Data

Dealing with malware instant growth in volume and variety become more and more challenging not only for industry area but also for academic research. Malware researchers frequently seek malware samples to analyze threat techniques and develop defenses. For the purposes of malware statics, analysis and to catch up with its trend, many AV vendors and malware collecting systems try to gather the released malware.

While collecting the malware through the Internet to the collecting servers, some of them might not be transferred successfully to the servers since network interruptions could happen, unfortunately. We named such kind of incompletely collected files as garbage data. Therefore, the malware databases have been collecting garbage files unintentionally while collecting a significant

amount of malware samples.

In this research, we figure out how to filter the garbage data from the malware samples. We performed clustering analysis as a first step of the data refining process. Garbage finding process will be carried out after obtaining the separated groups of malware samples. Clustering can be done by different algorithms which vary in the way of determining the difference or similarity among the data samples to formed the clusters and how proficiently they can find those clusters. We will be using the well-known k-means algorithm for clustering. Along with k-means, other clustering algorithms such as spectral clustering, hierarchical clustering, density-based spatial clustering (DBSCAN) and balanced iterative reducing and clustering using hierarchies (Birch) will be tested on large malware data set. To be able to handle the massive amount of data samples, we upgraded the simple k-means algorithm into nested k-means. The results obtained from these clustering techniques will be compared on scalability, performance and accuracy validations.

2.4 Binary Pattern Matching

Pattern matching is a conventional and existing problem in the field of computer science. It is one of the most basic mechanisms that supports various programming languages. Various real-world applications make use of pattern matching algorithm as a key role in their tasks. The patterns are generally in the shape of either a sequence or tree structure. For the sequence patterns, string matching involves as a one-dimensional pattern matching.

Typical pattern matching of binary strings is checking and locating the occurrence of one pattern, g built over a binary alphabet in another larger binary string, m, in which each character in both g and m is represented by a single bit. Unlike pattern recognition, the match has to be exactly the same in pattern matching.

In case of searching garbage from malware data set, counting the occurrence of g inside m is not necessary as the garbage file is just an incomplete file or prefix part of a malware file. Therefore, we just need to compare g and m, each bit by bit. Let g has a shorter length by bit then m. Then, matching procedure starts from comparing the first bit of each binary string and continue the matching process till the last bit of the g matched with the current bit, but not the last bit of the m. If g has the exact same bits as m, g is decided as a garbage file. But if the current bit of each file does not match with each other anymore while comparing process, we assume that both of them are not garbage files. The binary pattern matching algorithm is implemented using python language.

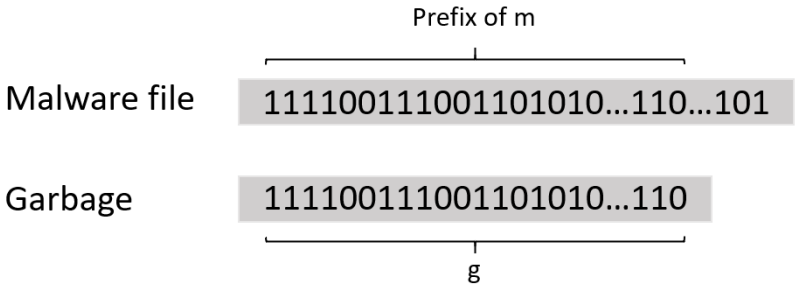

Figure 2.2: A garbage comparison with relative malware

Algorithm 1 describes the workflow of pattern matching procedure for binary files. The algorithm takes binary files as input and sets of malware files and garbage files. Finding garbage from binary files consists of two main steps. First, the input data set is sorted in ascending order. Rather than pattern matching the binary files with various sizes, making the files in order and comparing them later help in accelerating the matching performance. The binary files are sorted by the length size using quick sort algorithm, whose worst-time complexity is $O(n^2)$ but average-case complexity is O(nlogn). Besides quick sort, we also tried the merge sort algorithm with worst-case complexity of O(nlogn). Even though Quicksort has $O(n^2)$ in the worst case, that can happen only when the elements are already sorted in ascending or descending order, which is a very rare case in malware data set. As the innermost loop of quick sort algorithm is simpler, it can also get about 2 to 3 times faster than the merge sort. Moreover, quick sort does not use additional storage space to perform sorting while merge sort requires a temporary array to merge the final sorted arrays. More importantly, sorting binary files with quick sort practically faster than with merge sort. These reasons drive the decision to use quick sort in sorting the data set before the matching process. Binary pattern matching takes place as a second step. Since the string pattern is just the binary alphabet, we use exclusive or (xor) binary operation for comparing each bit of two files. According to the xor logical rules, if any of the resulting bits evaluates as 1, matching process stops and none of the files are assumed as garbage. If there is no resulting bit as 1 till the end of the shorter file, we marked that shorter file as garbage. The algorithm for matching two binary files is described in algorithm 2.

Algorithm 1: The algorithm for finding garbage from malware data set

Input : S - Binary data set
Output: M – Malware data set, G – Garbage data set
$Q \leftarrow QuickSort(S)$; $\#Q[i] \geq Q[j]\ \forall i < j$ $temp \leftarrow S[0]$;
$M \leftarrow [temp]$;
$G \leftarrow []$;
for $i \leftarrow 1$ **to** $length(Q) - 1$ **do**
    $res \leftarrow Matching(Q[i], temp)$;
    if $res\ is\ -1$ **then**
        $G.append(Q[i], temp)$;
        $continue$;
    else
        $S'.append(Q[i])$;
        $temp \leftarrow Q[i]$;
    end
end
return M, G;

Algorithm 2: The algorithm for comparing two binary files

Input : b,b' – binary files
Output: The result of matching
$buf_1 \leftarrow bytearray(b.read())$;
$buf_2 \leftarrow bytearray(b'.read())$;
$min\_length \leftarrow minlength(buf_1), length(buf_2)$;
for $i \leftarrow 0$ **to** min_length **do**
    if $buf_1[i]\ xor\ buf_2[i] > 0$ **then**
        $break$;
    end
end
if $i < min\_length$ **then**
    return 0;
else if $i < length(buf_1)$ **then**
    return 1;
else
    return -1;
end

2.5 Collected Garbage Data

For the purpose of practical malware analysis, there are some free sources that provide malware samples. Malware researchers make contributions with some malware sources to analyze the attacks and techniques. In this research, we collected the data from VirusShare.com website[5], a malware repository collecting, indexing, and freely sharing samples of malware to analysts, researchers, and the information security community.

2.5.1 Dataset

We collected 30 malware data sets from Virusshare, in which some of them contain 131,072 samples and some have 65536 samples in it. These data sets are listed in table 2.1, expressing their size, the number of files in each data set with the time added to the Virusshare database. To evaluate the scalability for the algorithms, we enlarge the data sets by making the combinations of two to five collected data sets, presented in table 2.2

Recently, a survey for IoT malware detection with deep learning approaches had been conducted over 15,000 malware samples supplied by Yokohama National University[6]. For this research, we also received 55,763 malware samples collected with IoTPOT[2], by Yokohama National University.

2.5.2 Result

First, we simply tried the binary pattern matching algorithm by oneself to know the garbage percentage of each collected and combined malware data set. The average percentage of garbage composition for a data set is around 4%, ranging from 1C to 8%. Execution time varies from one data set to another depending on the size of the data set. IoT malware data set contains a significate amount of garbage in it, which is more than 40% of the data set. Table 2.3 shows the average execution time and average garbage composition percentage of malware data sets with the different number of files inside.

Table 2.1: Collected malware data sets from VirusShare

Data ID	No. of Files	Size	Uploaded Time
VirusShare_00000	131,072	21.6 GB	2012-06-15
VirusShare_00002	131,072	59.1 GB	2012-06-16
VirusShare_00003	131,072	39.8 GB	2012-06-16
VirusShare_00004	131,072	30.6 GB	2012-06-16
VirusShare_00005	131,072	34.9 GB	2012-06-17
VirusShare_00006	131,072	53.8 GB	2012-06-29
VirusShare_00007	131,072	66.3 GB	2012-07-06
VirusShare_00008	131,072	75.9 GB	2012-07-07
VirusShare_00009	131,072	64.1 GB	2012-07-14
VirusShare_00010	131,072	45.3 GB	2012-09-15
VirusShare_00011	131,072	38.1 GB	2012-09-22
VirusShare_00012	131,072	37.4 GB	2012-09-25
VirusShare_00013	131,072	28.2 GB	2012-10-05
VirusShare_00014	131,072	38.7 GB	2012-10-11
VirusShare_00015	131,072	37.9 GB	2012-10-20
VirusShare_00016	131,072	38.8 GB	2012-10-25
VirusShare_00017	131,072	45.5 GB	2012-10-25
VirusShare_00018	131,072	39.5 GB	2012-10-29
VirusShare_00019	131,072	37.3 GB	2012-10-31
VirusShare_00020	131,072	55.5 GB	2012-11-05
VirusShare_00021	131,072	75.4GB	2012-11-20
VirusShare_00022	131,072	77.7 GB	2012-11-27
VirusShare_00323	65,536	23.6 GB	2018-06-30
VirusShare_00350	65,536	22.4 GB	2019-01-20
VirusShare_00351	65,536	18.3 GB	2019-01-20
VirusShare_00352	65,536	13.7 GB	2019-01-20
VirusShare_00353	65,536	15.5 GB	2019-01-29
VirusShare_00354	65,536	15.5 GB	2019-01-31
VirusShare_00355	65,536	12.8 GB	2019-02-08
VirusShare_00356	65,536	16.5 GB	2019-02-11

Table 2.2: The combinations of Virusshare data sets

No. of Combining Datasets	No. of Files	Data ID
2	262,144	VirusShare_00002_00003 VirusShare_00003_00004 VirusShare_00004_00005 VirusShare_00005_00006 VirusShare_00007_00008 VirusShare_00011_00012 VirusShare_00015_00016 VirusShare_00016_00017
3	393,216	VirusShare_00004_00005_00006 VirusShare_00007_00008_00009 VirusShare_00011_00012_00013 VirusShare_00015_00016_00017
4	524,288	VirusShare_00006_00007 _00008_00009 VirusShare_00011_00012 _00013_00014 VirusShare_00015_00016 _00017_00018
5	655,360	VirusShare_00015_00016_00017 _00018_00019

Table 2.3: Average garbage percentage and execution time of different data sets

No.of Files	Avg. File Size	Avg. Execution Time	Avg. Garbage (%)
55,763 (IoT)	6.2 GB	1.17hrs	41.3%
131,072	40 GB	2.75 hrs	2.8%
262,144	88 GB	9.50 hrs	2.5%
393,216	146 GB	15.75 hrs	4.0%
524,288	191 GB	18.45 hrs	4.8%
655,360	199 GB	20.00 hrs	5.4%

Chapter 3

Clustering Techniques

Clustering is a task of gathering the data points into a number of groups based on their similarity such that the data points in each group are more similar to each other than the data points from other groups. The resulting groups having similar data points are called clusters. The major difference from other machine learning analyses such as classification and regression is the type of data set used to train the models. This fact also defines whether the machine learning technique is a supervised or unsupervised approach. While classification or regression model uses a data set which contains data points tagged with associated class labels (supervised learning), clustering only required the data set having data points without provided with the labels (unsupervised learning).

To find the most suitable clustering technique, we compare five different algorithms: k-means, spectral clustering, hierarchical clustering, density-based spatial clustering (DBSCAN) and balanced iterative reducing and clustering using hierarchies (Birch). They can be distinguished into three categories[7]:

1. Partitioning
2. Hierarchical
3. Density-based

Partitioning algorithms separate the data set into the specified number of clusters based on the similarity or distance among the data samples. Hierarchical algorithms compose the clusters in the hierarchical structure. Density-based algorithms find the dense regions among data samples to form clusters and low-density regions create boundaries between the clusters.

In this chapter, we provide the detail of the clustering algorithms. For implementing the clustering algorithms, we referred the source code from

Keras and Scikit-learn frameworks. Since the data samples we used are unlabeled malware files, we choose the unsupervised learning algorithms that are known to be working well with large data set.

3.1 K-means Algorithm

K-means clustering is one of the simplest and frequently used unsupervised learning algorithms, especially in data mining and statistics. Being a partitioning algorithm, its goal is to form groups of data points based on the number of clusters, represented by the variable k. K needs to be predefined before the execution. K-means uses an iterative refinement method to produce its final clustering based on the number of clusters defined by the user and the data set. Initially, k-means randomly chooses k as the mean values of k clusters, called centroids, and find the nearest data points of the chosen centroids to form k clusters. Then, it iteratively recalculates the new centroids for each cluster until the algorithm converges to one optimum value. K-means clustering would be suited with the numerical data with a low dimensionality because numerical data is used to compute the mean value. The type of data best suited for K-Means clustering would be numerical data with a relatively lower number of dimensions. The algorithm works as follow:

1. K points are randomly initialized as centroids of clusters based on the predefined value of k.

2. To form the k clusters, every data points of the data set are assigned to the nearest centroid by the distance.

   The Euclidean distance is used to in calculating the distance between each data points and the initialized centroids. Although there are many other metrics to find the closest distance, we apply Euclidean distance because several previous research about clustering analysis gained great outcomes using the Euclidean distance.

3. The centroids are recalculated by averaging all of the data points assigned in each cluster so that the total intra-cluster variance can be reduced.

4. Step 2 and 3 iterate until some criteria is met.

   Criteria are normally, when there are no changes in the centroids' values, the sum of distances between the data points and the centroid of each cluster does not change anymore, the data points assigned to the

clusters are the same as the previous assignment or the maximum iteration number has reached in case the algorithm is given a fixed iteration times[8].

Advantages

- Since k-means is a simple clustering algorithm, it can be implemented easily.
- K-means has only a few computations, only computing and comparing distance among data points and grouping clusters. Thus, it can be computationally faster than hierarchical clustering, having the time complexity of O(n), where n is the number of data samples.
- It can scale up to large data set.
- Additionally, It can also easily adapts to new data samples.

Disadvantages

- The number of clusters, k has to be specified manually.
- The clustering results can vary depending on initial values. K-means also randomly select the initial centroids for k clusters. Therefore, the results can be different from one execution and another, lacking inconsistency.
- K-means has difficulty with clustering data sets of varying sizes and density.
- K-means cannot identify outliers. The outliers or noises of the data set can effect in clustering process as the cluster might drag the outliers in or outliers can themselves becomes a cluster [9].

3.2 Hierarchical Clustering

Hierarchical clustering algorithms seek to build a hierarchy of cluster. It works well for the data set with nested clusters, eg. geometrical data. It starts with some initial clusters and gradually converge to the solution. Hierarchical clustering has two categories: agglomerative and divisive. The agglomerative approach initially takes each data point as an individual cluster and the iteratively merge the clusters until the final cluster contains all data points

in it. According to how this approach merges the clusters, it is also called a bottom-up approach. As an opposite technique of agglomerative clustering, divisive clustering techniques follow top-down flow which starts from a single cluster having all data points in it and iteratively split the cluster into smaller ones until each cluster contains one data point. For our research, we use agglomerative approach for clustering malware data set.

Agglomerative hierarchical clustering algorithm includes the following steps:

1. As an initial step, the algorithm takes each data point as a single cluster and we decide a specific proximity matrix to determine the distance between the clusters.

   There are four distance functions available for proximity matrix: single linkage (min), average linkage, complete linkage and ward (max)[10]. Single linkage means the distance between two clusters is defined as the minimum distance between one point of the first cluster and another point of the second cluster. Complete linkage takes a maximum distance of two data points value as the distance between two clusters. Average linkage calculates the distance of all data points from the first cluster with all others from the second cluster and takes the average distance as the distance between the clusters. Ward is similar to average linkage except that it uses the sum of squares to calculate the distance between the points. In this research, we use ward as a distance function.

2. To find the closest pair of clusters, it computes the similarity (distance) between each of the clusters.

3. Then, the similar clusters are merged to form a cluster according to the distance function.

4. Iteration through step 2 and 3 continues until all data points are merged into one last cluster.

In general, hierarchical clustering is forming a single tree of clusters where each node is representing the clusters and each data point starts as a tree leaf. The root of the tree is the final cluster containing all of the data points (Figure 3.2) shows how hierarchical clustering cut out the k clusters from the final cluster (complete tree). In the figure, the algorithm successively forms a single tree of clusters and then cut at a certain level k, resulting in 4 clusters.

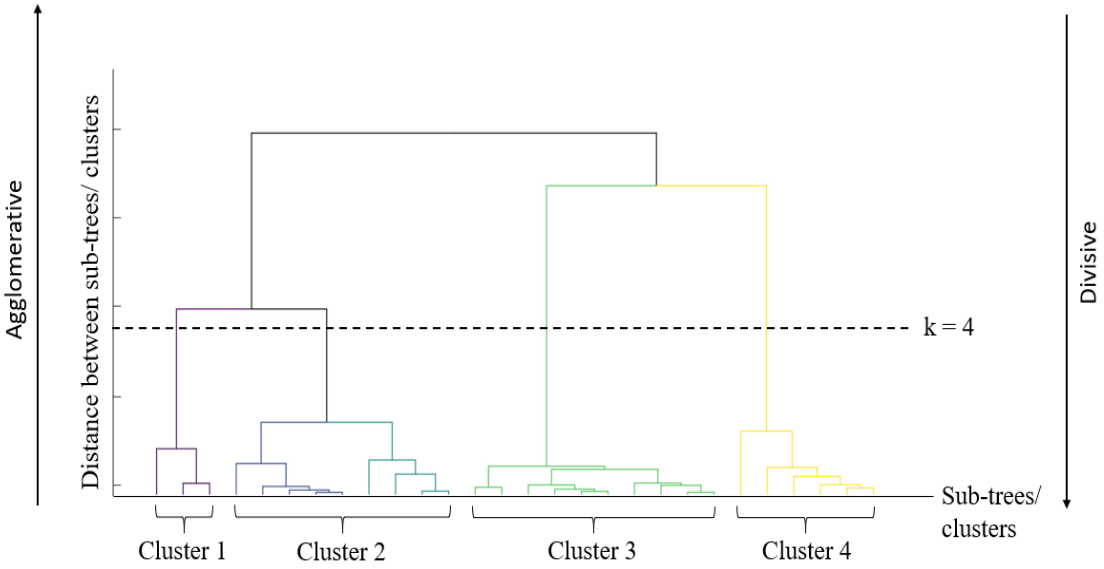

Figure 3.1: A dendrogram representing the clustering technique of hierarchical clustering algorithm

Advantages

- The number of clusters is not necessarily required to be specified.
- Like k-means, hierarchical clustering algorithms are easy to implement.
- It can output the hierarchical structure of a cluster tree(dendrogram), which can help in deciding the number of clusters.

Disadvantages

- The main drawback of hierarchical clustering is its time complexity. Comparing with other algorithms, it has a relatively higher complexity of $O(n^2 \log n)$, n being the number of data points.
- There are no backtrackings which mean once one cluster is created, the membership data points cannot be moved around.
- Depending on the choice of the distance matrix, it can be sensitive to noises and outliers. Besides, it can face difficulty in handling different sized and convex-shaped clusters[11].

3.3 Density-based Spatial Clustering (DBSCAN)

One clustering technique that does not require the specification of the number of clusters is DBSCAN. However, DBSCAN basically requires two parameters: eps and min_samples. Eps specifies the maximum distance between two data samples in which one of them is supposed to be a neighborhood of another which is a core point of a cluster. Min_samples defines the minimum numbers of samples that have to be in the neighborhood together with the core sample. It assumes that a cluster is a dense region with data points that is greater than min_samples within the range of eps of the core point and each cluster is separated from another by lower density.

The steps in the DBSCAN algorithm includes:

1. It starts with an arbitrary starting data point that has not been visited. The neighborhood of this point is extracted using the maximum distance eps.

2. If its neighborhood contains the sufficient number of points according to min_samples, the clustering begins. That starting point becomes the core point of the cluster as well. Otherwise, the point is assumed as noise. Later on, this point can probably be a in the neighborhood of other point and hence be a part of a cluster. In either case, this point is marked as a visited point.

3. The points in the neighborhood of the core point are then used to search their respective neighborhood points since these points are still new un-visited points.

4. This process of steps 2 and 3 is repeated until all points in the cluster have been visited and labeled forming the density-connected cluster.

5. Then, the new un-visited point in the data set is retrieved and the algorithm repeats through step 1 to 4 until all points have been visited and become either noise or part of a cluster[12].

Advantages

- DBSCAN performs well with the data set insisting of high-density clusters versus low-density clusters.

- It is resistant to noise and can handle outliers of the data set.

- It can also handle different shaped and sized clusters.

– Unlike k-means, the number of clusters does not need to be defined in advance.

Although DBSCAN has an advantage of not requiring the pre-defined value for the number of clusters, the distance value, eps becomes challenging to estimate. Especially when the clusters have varying density. As it can separately define high-density clusters from low-density clusters, it performs well for the case like crime Incident or crime rate statistics.

Disadvantages

– Although DBSCAN can separate the high-density clusters from low-density clusters, it does not work well with clusters of varying densities or similar density.

– Also, it cannot handle high dimensional data well[13].

3.4 Spectral Clustering

3.5 Spectral Clustering

Spectral clustering makes use of the k-means algorithm as a step of its algorithm. Lately, it becomes popular due to its simple implementation and great performance with graph-based clustering. Three main steps to perform spectral clustering are as follows:

1. First, it creates a similarity graph for all data points.

   There are different ways to construct a similarity graph such as ε-neighborhood graph, k-nearest neighbor graphs, and a fully connected graph. Among them, we use k-nearest neighbor graphs in this research. Each data point in the data set is represented as a vertex. When one vertex is among the k-nearest neighbor of another vertex, it is assumed that these vertices are connected. The resulting graph is known as the k-nearest neighbor graph. After connecting the associated vertices, the weights of the connecting edges are added by the similarity of their endpoints or by using 0-1 weight[14].

2. From the resulting similarity graph, the associated Laplacian matrix is computed by subtracting the weight matrix from the (diagonal) degree matrix. Then, compute the eigenvectors of the Laplacian matrix to define a feature vector for each object.

3. Finally, clustering algorithm k-means is applied with these features to separate objects into clusters[15].

Advantages

- In k-means case, the membership data points can be assumed in a spherical area as the centroid is normally at the center of the cluster. Spectral Clustering does not make such kind of strong assumption and can cluster the data more accurately.
- It gives better results than other algorithms.

Disadvantages

- As it uses k-means algorithm for clustering in the last step, clustering results may vary depending on the initial centroids.
- For large data sets, the time complexity can be computationally high since the algorithm has to compute eigenvalues and eigenvectors and perform clustering over these eigenvectors[16].

3.6 Balanced Iterative Reducing and Clustering using Hierarchies (Birch)

Birch is a multiphase clustering algorithm used to perform hierarchical clustering over large data sets. It builds a clustering feature (CF) tree, which is a hierarchical data structure for multiphase clustering. This algorithm is often mentioned as two-step clustering because it consists of two main steps. These steps include:

1. Birch scans the data set to build an initial CF tree and stores the clustering features of the data.

   CF tree is a height-balanced tree in which the leaf node stores the cluster of data points in the form of clustering features (CF). Each CF is represented by a tuple of three numbers (N, LS, SS).

   Given n-dimentional data points in the cluster,
   CF = (N,LS,SS)
   N = the number of data points in the cluster,
   LS = the linear sum of the data points in the cluster,

SS = the squared sum of the data points in the cluster,

Non-leaf node or parent node stores the tuple of CF summations of its child nodes. CF = (CF1 + CF2 + ... + CFN), where
CF = ((N1 + N2 + ... +NN), (LS1 + LS2 + ... +LSN), (SS1 + SS2 + ... +SSN))
CF = clustering feature stored in the parent node, resulting from the summations of child clustering features
CF1, CF2, ..., CFN = clustering features stored in the child node
Brich algorithm requires two paramenters: branching factor (B) and threshold (T). Branching factor specifies the maximum number of sub-clusters in each non-leaf node. When new data points are scanned and the number of sub-clusters exceeds the value of branching factor, the node is split into two nodes owning respective sub-clusters in each. The radius of the sub-cluster should be less than the threshold value. Otherwise, a new sub-cluster is created. Birch algorithm makes full use of available memory to constructs the hierarchical CF tree with clusters of data features while minimizing I/O cost. To avoid running out of memory, the threshold can be adjusted by increasing the value. However, it is hard to figure out a suitable threshold value.

2. Then, Birch applies the existing clustering algorithm on the leaf nodes of CF tree[17].

(Figure 2.2) shows the basic structure of hierarchical CF tree and relations among leaf nodes, non-leaf nodes, and the root node.

Advantages

- Birch can make good clustering with a single scan of the database
- Same as k-means, it has linear time complexity, O(n) where n is the number of data samples.

Disadvantages

- Like k-means, it prefers spherical shaped and similar sized data clusters as it uses the threshold value to limit the cluster boundary[18].

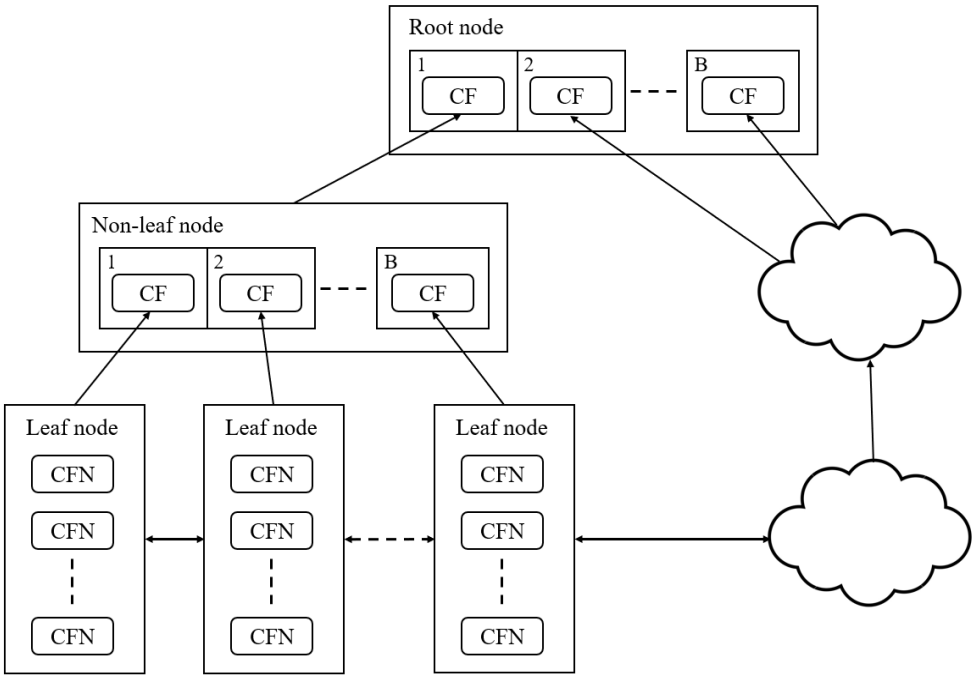

Figure 3.2: Basic structure of hierarchical CF tree

3.7 Space and Time Complexity

The following table 3.1 shows the comparison of five different clustering algorithms in terms of space and time complexity.

Table 3.1: Space and time complexity of different clustering algorithms

Clustering Type	Algorithm	Time Complexity	Space Complexity
Partitioning	K-means	O(n)	O(n)
	Birch	O(n)	O(n)
Hierarchical	Agglomerative	$O(n^2 \log n))$	$O(n^2)$
	DBSCAN	$O(n^2)$	$O(n^2)$
Density-based	Spectral	$O(n^3)$	$O(n^2)$

Chapter 4

Comparison of Clustering Algorithms

In this chapter, we make a comparison of several clustering algorithms mentioned in the previous chapter. Aiming for this research, the required abilities of the clustering algorithms to be able to handle large are the scalability, execution time and performance (how efficiently the algorithm can make partitions over the data set). Each algorithm has its own strength and drawback as described in the previous chapter. All of the algorithms have limitations due to some specific conditions of data or the parameter specifications of the algorithm. Theoretically, it is difficult to assume which algorithm is better than which algorithm or which one is the best of all. Therefore, we make practical experiments over the malware database to find out the most suitable clustering algorithm for our research.

Rather than just experimenting their clustering performance of these five algorithms, we combine each algorithm with binary pattern matching algorithm, so that overall performance can be assessed. The accuracy will be evaluated by comparing the final results (number of garbage) with the result get by using only binary pattern matching.

4.1 Experiments on Malware

For the first experiment, we set the number of clusters as 30 for the algorithms (k-means, agglomerative hierarchical clustering, spectral clustering, and birch) which required pre-defined cluster value. We fixed the feature size as 512 for each binary file to feed as input to the clustering model. For other parameters, we mostly used the default value selected by the Scikit-learn libraries for clustering algorithms.

To verify the performance of the combination of clustering algorithms and binary pattern matching algorithm, we conduct the experiments on ten data sets among the downloaded malware data sets from Virusshare malware database. Accuracy is simply computed based on the total amount of garbage found by using binary pattern matching algorithm only.

$$\text{Accuracy} = \frac{\text{no. of garbage found by BPM}}{\text{no. of garbage found by (Clustering + BPM)}}$$

Among them, eight malware data sets: VirusShare_00350, VirusShare_00351, VirusShare_00352, VirusShare_00353, VirusShare_00354, VirusShare_00355 and VirusShare_00356 include 65536 malware files and the average size of data set is 19 GB. The results of executing clustering algorithms together with binary pattern matching are expressed in table 4.1. The other sets: VirusShare_00002, VirusSharee_00003, and VirusShare_00009, have 131072 malware files in each and the average size of the data set is 50 GB. Table 4.2 shows the executed results over these data sets.

All experiments are performed on Ubuntu 18.4 using 2 GPU: GeForce RTX 2080 and GeForce GTX 780. The host system itself is running a Ubuntu 18.4 installation powered by an Intel i7-4770 K Core(TM) 3.50 GHz CPU and 32GB of RAM.

4.2 Result

According to the results from table 4.1, hierarchical clustering and spectral clustering take the longest time for execution. The other algorithms got the similar execution time. Regarding the accuracy comparison, spectral clustering has the lowest accuracy. spectral clustering mostly creates the data clusters with unbalanced files. Some clusters contain more than half of the total files of data set while others only contain very few files. This clustering behavior can probably decrease the accuracy of the algorithm.

In table 4.2, we can see the difference between the performance of the algorithms. Birch cannot handle the data set with 131072 files. Since it uses all available memory space, memory error occurs during processing the algorithm. We tried with higher threshold values but it can still not cluster the data set. Spectral clustering has the best accuracy among all but it also takes too much time for execution. Although DBSCAN is as fast as k-means, its accuracy is the lowest while other algorithms get nearly 100%. The results from both tables show that k-means is the fastest algorithm with great accuracy. Because of these facts, we choose the k-means algorithm for clustering process for this research.

Table 4.1: Comparison of different clustering algorithms in terms of execution time and accuracy (65536 files)

Algorithm (+ BPM)	Avg. Execution Time	Avg. Accuracy
K-means	12 min	100%
Birch	15 min	100%
Hierarchical Clustering	1 hr	100%
DBSCAN	16 min	80.6%
Spectral Clustering	2 hrs	100%

Table 4.2: Comparison of different clustering algorithms in terms of execution time and accuracy (131072 files)

Algorithm (+ BPM)	Avg. Execution Time	Avg. Accuracy
K-means	2 hrs	99.9%
Birch	-	-
Hierarchical Clustering	4 hrs	99.8%
DBSCAN	2 hrs	56.7%
Spectral Clustering	8 hrs	100%

Chapter 5

Parameter Specification

Based on the experimental results from the previous chapter, k-means algorithm is chosen to do clustering the malware data set. Applying k-means requires some parameters to be specified overhead. In this chapter, we will describe the parameter specifications for the k-means algorithm

5.1 Feature Size

In the process of matching the binary files, the algorithm requires the whole data information of binary files to compare one file with another. However, if we feed the whole malware files to clustering algorithm, it will make the processing time extremely long which is not good for dealing with huge data set. Moreover, more data feature does not always tend to better accuracy, it can also make the model confuse in partitioning the data set. Therefore, we find out what data size for each malware file would be enough for k-means clustering. We tested on IoT malware dataset with the data size of 512, 1024, 2048, and 4096 setting the number of clusters, k from 10 to 100. These data sizes are compared with each other in terms of accuracy and execution time. Then, the comparison of how these two data size can affect the clustering performance is carried out to select the better one.

In figure (5.1), it can be seen that the execution time decrease gradually with the increase in cluster size for 512 data size. There is also an apparent decrease in execution time at the cluster value 30. In 4096 data size, the decreasing execution time is unstable and every clustering takes more time than the data size of 512. Here, we focus on the two sizes: 512 and 4096 to make the difference more clear, as the differences among 512, 1024, 2048, and 4096 data sizes grow slightly with the increment of data size.

To get a better choice of the input data size, we also compare the accuracy

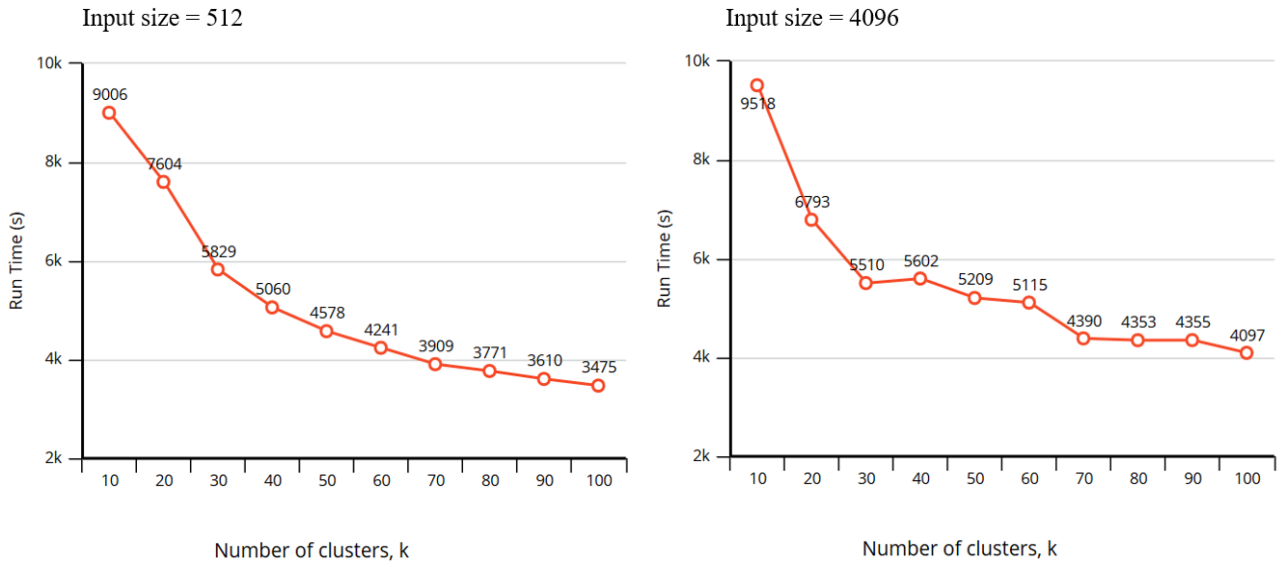

Figure 5.1: Comparison of data sizes 512 and 4096 in terms of execution time

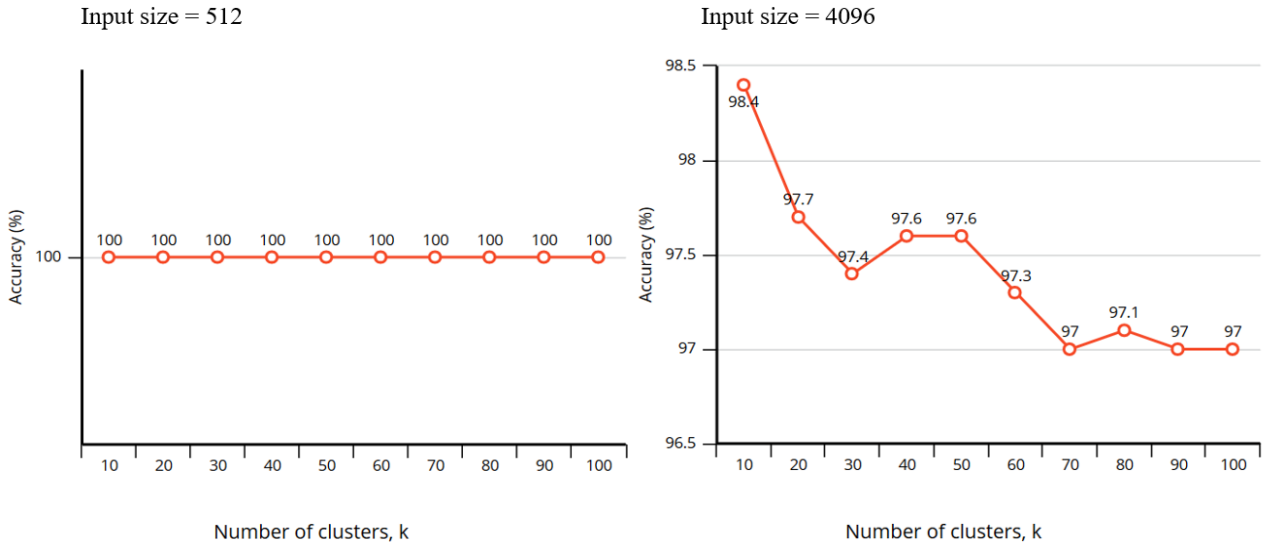

Figure 5.2: Comparison of data sizes 512 and 4096 in terms of accuracy

of how precisely the algorithm can make clusters with each data size. First, we make clustering using different data sizes and then find the garbage file with binary pattern matching in each cluster. Again, we take the result of binary pattern matching itself to compute the accuracy of garbage finding after clustering. In figure (5.2), the accuracy with 4096 is not as good as 512 data size, even though the results are closed to 100%. According to the results of comparing different input sizes, we decide to use 512, which gives the highest accuracy with the lowest execution time, as an input size of k-means clustering algorithm.

5.2 Optimal Number of Clusters, K

Being one of the partitioning methods, k-means algorithm requires a predefined k, the number of clusters. The number of clusters should be determined appropriately as they can affect the clustering result. There is no standard answer for how correct is the chosen number of clusters. Different shaped and sized data sets have different appropriate k value. To select the optimal k, we apply the elbow method. It can be said to be the most well-known method which gives a visual measure to find the best pick for the value of k.

Elbow method measures the sum of squared errors for different numbers of clusters. The sum of squared errors means the sum of the squared distance of each data points from its centroid of a cluster. Just a k-means, we use Euclidean Distance as a distance metric. After plotting the sum of squares at each number of clusters matched with the respective number of clusters, we can see a point with a slope from steep to shallow, decreasing in error sum. That point is an elbow point and it determines the optimal number of clusters[19].

In figure (5.3), the bend indicates that the bigger number of clusters beyond the third have small decreases in error sum, pointing that the optimal number of clusters is 30. We used the IoT malware data set, VirusShare_00350 to VirusShare_00356 to find the optimal k for each data set, most of the results point out the optimal k around 30. Therefore, we choose value 30 for the number of clusters for the clustering process in our further experiments.

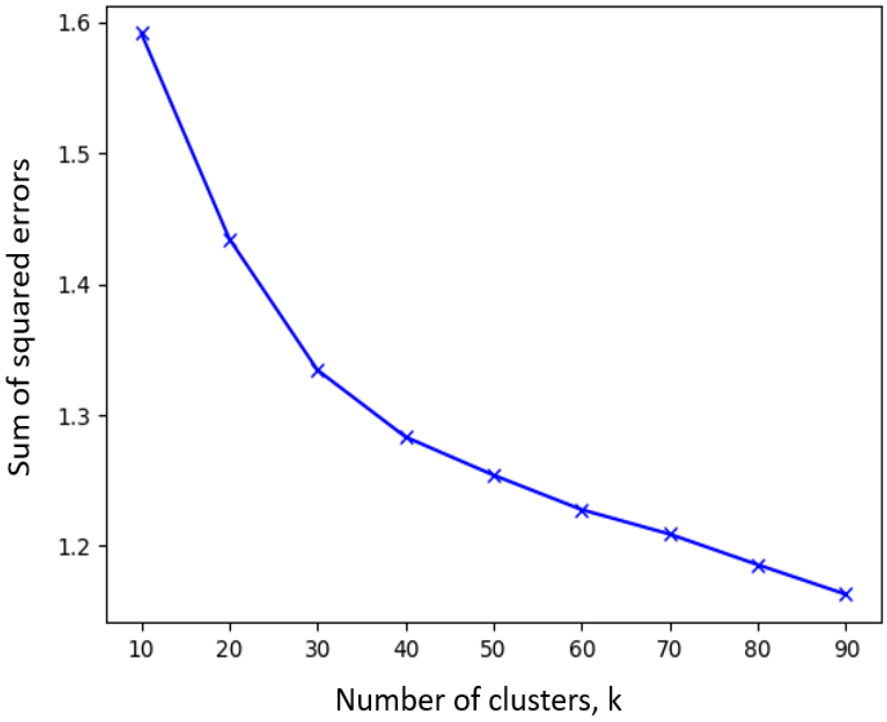

Figure 5.3: The elbow method showing the optimal k

Chapter 6

Nested Clustering

In our previous chapters, we have made practical experiments to find the most suitable clustering algorithm along with its optimal parameters. K-means performs well with binary pattern matching in searching garbage from malware samples. Although we tested the algorithms on multiple malware data sets, k-means gives the high stable accuracy and fastest execution time among five different clustering algorithms.

However, our aim of this research is not only just finding the garbage from the data set but also making the algorithm as fast as possible. As we have seen in the table 2.3, the amount of garbage in the malware data set increases when we collect more malware files a data set. Thus, the faster the algorithm, the more efficient the algorithm so that we can find as much garbage as possible in a malware database.

We speed up the binary pattern matching algorithm by initially sorting the data set using quick-sort before matching the binary files. As for the clustering process, we modify the ordinary k-means algorithm by iterative clustering. We have experienced that some algorithms like DBSCAN, make the unbalanced clusters, that is the number of data points in a cluster is not equal with the others. This can lead to low accuracy in finding garbage. Although k-means can cluster better than other algorithms, there is still some unbalanced clusters in the result. Again, the purpose of clustering the malware files before the matching process is to help the matching algorithm find the garbage easier and faster. Instead of comparing one file with the whole malware data set, it is better to compare only with similar files within the cluster. If the clusters are not well-balanced in the number of files, the clustering process cannot give great help to the binary pattern matching stage. Therefore, we created the nested clustering algorithm using k-means.

6.1 Nested K-means Algorithm

For nested k-means algorithm, we need to specify two parameters: the number of clusters (k) and the maximum number of files in each cluster (f).

The algorithm includes three parts.

1. First, the data set is separated into k clusters once with k-means algorithm.

2. The second step entirely depends on the results gained from the first step. If there is a cluster that has more file than f, we execute k-means again with the new number of clusters. New k will be defined by dividing the total number of files in the current cluster by f. After the k-means clustering with new k, we then check again whether is there any cluster exceeding the maximum number of files in each cluster. If so, k-means will be executed again. Step 2 will iterate until there are no unbalanced clusters, which means clusters exceeding the maximum number of files, f.

   However, there are also some clusters that cannot be partitioned anymore despite the second trial of k-means. In that case, we do not iterate k-means clustering for such kind of clusters again. Forced clustering might lead to low accuracy in the file matching process.

3. Finally, binary pattern matching is used to find garbage from the final clusters resulting from the nested k-means clustering.

   The clustering and garbage filtering procedures are shown in the pseudocode in the algorithm 3.

Algorithm 3: Nested k-means algorithm for finding garbage from malware data set

Input : S - Binary data set, k – number of clusters, f – maximum number of files

Output: M – Malware data set, G – Garbage data set

```
C ← Kmeans(S, k); #K − list of clusters
i ← 0;
while i < len(C) do
    if len(C[i]) > f then
        k' ← (len(K[i]/f) + 1); #k − new number of clusters
        C' ← (S, k');;
        j = 0;
        while i < len(C') do
            if len(C'[i]) = 0 then
                del(C'[i]);
                j ← j − 1;
            end
            i ← i + 1;
        end
        if len(C') > 1 then
            C.extend(C');
            del(C[i]);
            i ← i − 1;
        end
        i ← i + 1;
    end
end
for i ← 0 to length(C) − 1 do
    M, G ← parallel_bpm(C[i]);
end
return M, G;
```

6.2 Dataset and Parameter Specification

Through our earlier experiments, we have learned the suitable parameters for k-means clustering algorithm. The input data size 512 takes less time in execution and the optimal number of clusters, k turns out to be 30 measured with the elbow method. K-means shows better performance than other algorithms using these parameters. As nested k-means is just an extended algorithm of the ordinary k-means by using it more than for iterative clustering, we stick with these parameter specifications. To make sure that these parameters can also bring great performance in nest k-means clustering, we made a few testings with other values. We increased the parameter values. The processing time increases and the accuracy decreases with the increase in the number of clusters and the input data size. Therefore, we proceed most of our execution with 512 for the input data and 30 as the number of clusters. When dealing with larger data sets with more than 262,144 samples, we set the number of clusters value as 50 to avoid having too many files in a cluster.

In this experiment, we used the data sets: Virusshare_00000 to VirusShare_00022, listed in table 2.1, having 131,072 samples in each and use every combined data sets, described in table 2.2, in a total of 38 data sets of varying numbers of files. To have a clear comparison, we find the average of the execution time and accuracy of the data sets of the same number of files respectively. Besides, we also used the IoT malware data set supplied by the Yokohama National University.

6.3 Result

Table 6.1: Performance comparison in terms of execution time and accuracy

No. of File	BPM	Nest K-means + BPM	
	Avg. Execution Time	Avg. Execution Time	Avg. Accuracy
55,763	1.17 hrs	0.75 hrs	100.0%
131,072	2.75 hrs	1.50 hrs	99.9%
262,144	9.50 hrs	3.75 hrs	98.9%
393,216	15.75 hrs	6.75 hrs	99.9%
524,288	18.45 hrs	8.30 hrs	99.9%
655,360	20.00 hrs	12.45 hrs	99.9%

Table 6.1 present the performance comparison among binary pattern matching and pattern matching after clustering with nested k-means. Apparently,

nested k-means works fastest among all. It can still maintain the high accuracy of found garbage. The resulting clusters and the number of files in each cluster varies depending on the total number of files included in the set, shown in table 6.3 Clustering cannot be guaranteed that every garbage will be in the same cluster with its related malware file. Therefore, some garbage files are failed to be detected when clustering is used as a a preprocessing step of pattern matching. Table 6.2 shows the missed garbage files, related malware file and the clusters that includes the garbage and malware. As an example case, we used VirusShare_00000 to get that results. From the above analysis, we can conclude that nested k-means algorithm can accurately cluster the malware files, which speed up the pattern matching process two to three times faster.

Table 6.2: The missed garbage files of VirusShare_00000 data set

(Garbage, Cluster No. of Garbage)	(Malware, Cluster No. of Malware)
VirusShare_90ed09c271774df 3146af8ce8d6d8d52, C - 96	VirusShare_92d88b4eaa2213c 914a62a830285ad78, C - 82
VirusShare_fe5760b4a3eb895 b3039fc322f3dea0d, C - 92	VirusShare_7855867ed973953 0e698ffa63f3588d6, C - 102
VirusShare_d029410010484ea 900e1ece20a14f272, C - 55	VirusShare_085dfcef851c6b2 dc90502fc9b990071, C - 102
VirusShare_179ce69c89130af 6a0dc38796765d7d2, C - 55	VirusShare_bcfee9d55d58f376 61aae3a8fc7fa6f4, C - 99
VirusShare_44d88612fea8a8f 36de82e1278abb02f, C - 82	VirusShare_f23ddc28faac06f f61c7bd52ff76d6c7, C - 103
VirusShare_aa231800b145f2b d92f96e0509cc27c2, C - 55	VirusShare_c97b10a7c225710 323d4f5656c50cdd4, C - 102
VirusShare_a57e3ade32103e1 f21e27030d122eed9, C - 56	VirusShare_38996d70cebb9c9 243bd04bdc4f21761, C - 99
VirusShare_3fb4fc50717a00c 6d82a4cee79c9679b, C - 91	VirusShare_63c3ab4e6c12354 71ddf31f1e1de0f90, C - 106

Table 6.3: Resulting number of clusters and number of files in a cluster

No. of Files	No. of Cluster	No. of Files in Cluster
55,763 (IoT)	52	$400 \sim 1800$
131,072	$30 \sim 90$	$300 \sim 5000$
262,144	$80 \sim 90$	$400 \sim 5000$
393,216	$90 \sim 100$	$1000 \sim 6000$
524,288	$100 \sim 150$	$1000 \sim 9000$
655,360	134	$1000 \sim 15000$

Chapter 7

Related work and Conclusion

7.1 Related Work

In the literature, there are some nested clustering algorithms that have been proposed for various analyses. The approach proposed by Xia et al[20]. classifies the freeway operating condition into different flow phases. They apply the Bayesian Information Criterion (BIC) to determine the optimum number of clusters and use an agglomerative clustering algorithm. After grouping the traffic data into a specific number of clusters, the clustering process is repeated on all sub-clusters until the dissimilarity between the data points is not significant enough for further clustering. This technique is dedicated to performing effectively for data mining in a broad range of roadways analysis.

Li et al. [21] tried to detect nested clusters (clusters composed of sub-clusters) or clusters of multi-density (clusters formed in different densities) in a data set such as a geographical data set. This research discovers the hierarchical-structured clusters of nested data set. Agglomerative k-means is embedded in the generation of cluster tree at a different level of clustering. Then, cluster validation techniques are used to evaluate clusters generated at each level. Based on the evaluated result, the agglomerative k-means is iterated for the clusters with the nested structure or different densities.

These approaches perform nested clustering based on the cluster evaluation or the optimum number of clusters. As for our nesting k-means clustering, we try to reduce the unbalanced sub-clusters by iterative clustering, not depending on the number of clusters.

Furthermore, nested clustering approach is used to aid in the decision-making process of autonomous learning [22]. Instead of building a decision tree, this approach looks for a hierarchical structure of rules of execution. It applies the algorithm in a nested manner and a solution is driven when the

algorithm converges.

7.2 Conclusion and Future Work

This thesis presents our study on refining large malware data set, by separating the garbage (incomplete binary files) from the large data set. The faster the algorithm, the better in dealing with the large data sets in our case. Thus, we made the clustering process nested to reduce unbalanced clusters and use the advantage of quick sort to accelerate the matching process. By using the combination of the pattern matching algorithm and iterative clustering with simple machine learning method, we obtain the optimal results. Based on our experimental results, our approach gives high accuracy within a short time. We successfully found out almost every unnecessary garbage from the collected data sets from Virusshare and IoT malware data set.

The number of garbage composition gets higher with the increasing number of malware samples in the data set. In this research, we have made experiments over some malware folders of Virusshare and IoT malware set. Aiming to give a good aid to malware analyses, We expect the current system can be modified to accelerate the process to be able to deal with bigger data sets. Despite the excessive growth rate of malware, the method for automatic and labeling of new malware samples is still in lack. We hope to make use of our approach as a preprocessing step of malware families identification or malware clustering.

Bibliography

[1] V. Harrison and J. Pagliery. Nearly 1 million new malware threats released every day, (2019, Apr 14), https://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/

[2] J. Su, V. Danilo Vasconcellos, S. Prasad, S. Daniele, Y. Feng and K. Sakurai. Lightweight Classification of IoT Malware based on ImageRecognition, *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pp. 664-669, Tokyo, (2018)

[3] Cisco 2018 Annual Cybersecurity Report, https://www.cisco.com/c/m/en_au/products/security/offers/annual-cybersecurity-report-2018.html

[4] B. Rankin. A Brief History of Malware, (2018, Apr 5), https://www.lastline.com/blog/history-of-malware-it-evolution-and-impact/

[5] Virusshare.com, https://virusshare.com/

[6] K. D. T. Nguyen, T. M. Tuan, S. H. Le, A. P. Viet, M. Ogawa and N. L. Minh. Comparison of Three Deep Learning-based Approaches for IoT Malware Detection, *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, pp. 382-388, Ho Chi Minh City, (2018)

[7] D. Sisodia, L. Singh and S. Sisodia. Clustering Techniques: A Brief Survey of Different Clustering Algorithms, (2012)

[8] D. L. Yse. A complete guide to K-means clustering algorithm, (2019, May), https://www.kdnuggets.com/2019/05/guide-k-means-clustering-algorithm.html

[9] K-Means Advantages and Disadvantages, https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages

[10] C. Reddy. Understanding the concept of Hierarchical clustering Technique, (2018, Dec 10), https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec

[11] A. Naik. Hierarchical clustering algorithm, https://sites.google.com/site/dataclusteringalgorithms/hierarchical-clustering-algorithm

[12] G. Seif. The 5 Clustering Algorithms Data Scientists Need to Know, (2018, Feb 6), https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec

[13] E. Lutins. DBSCAN, (2017, Sep 6), https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818

[14] J. Liu, Jiawei. *Data Clustering Algorithms and Applications (Ch-8)*, Chapman and Hall/CRC, (2014)

[15] A. Aoullay. Spectral Clustering for beginners, (2018, May 8), https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8

[16] N. Doshi. Spectral clustering, (2019, Feb 5), https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7

[17] T. Zhang and R. Ramakrishnan. Birch, https://slideplayer.com/slide/4224423/

[18] M. Kuchaki Rafsanjani, Z. Asghari Varzaneh and N. Emami Chukanlo, A survey of hierarchical clustering algorithms, *The Journal of Mathematics and Computer Science*, vol.5, no.3, pp.229-240 (2012)

[19] K. Mahendru. How to Determine the Optimal K for K-Means, (2019, Jun 17), https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb

[20] J. Xia and M. Chen. A Nested Clustering Technique for Freeway Operating Condition Classification, *Computer-aided Civil and Infrastructure Engineering*, vol.22, no.6, pp.430-437 (2007)

[21] Xutao Li,Yunming Ye,Mark Junjie Li and Michael K. Ng. On Cluster Tree for Nested and Multi-Density Data Clustering, *Pattern Recognition*, vol.43, no.9, pp.3130-3143 (2010)

[22] James S. Albus and Alberto Lacaze and Alex Meystel. Algorithm of Nested Clustering for Unsupervised Learning, *Proceedings of Tenth International Symposium on Intelligent Control*, pp.197-202 (1995)