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

Yoon Myet Thwe, Mizuhito Ogawa
Japan Advanced Institute of Science and Technology (JAIST)
Nomi, Japan
{yoonmyetthwe96,mizuhito}@jaist.ac.jp

Pham Ngoc Dung
Le Quy Don Technical University
Ha Noi, Viet Nam
phamdunghc1@gmail.com

***Abstract*—Malware databases have been unintentionally collecting garbage (incomplete malware) together with malware through the Internet. This paper focuses on finding garbage (incomplete malware) from large malware datasets using binary pattern matching and speed up the matching by using nested clustering as a preprocessing. To verify the effectiveness of our method, we conduct experiments on various malware datasets. The results show that our method works efficiently while maintaining high accuracy.**

***Index Terms*—binary pattern matching, nested clustering, malware, incomplete malware**

I. INTRODUCTION

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. 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.

While collecting malware files through the network, unexpected interruptions like network failures may cut out the downloading process of a 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. For instance, the comparison of three deep learning approaches is performed on IoT malware dataset of 15,000 samples [1], in which one-third of it is garbage. Typical malware indexing is made by MD5, i.e, the MD5 hash function iteratively computes the hash value of a binary file from the top to the end and a proper prefix looks like a totally different file.

Since the proportions of collected garbage files are not small enough to be negligible, we try to filter the garbage from the malware samples. This research aimed to implement the method for refining the data set by detecting garbage among the mixture of malware with garbage files. We define each garbage is a proper 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 a quadratic manner. Instead, we investigate an application of clustering techniques as a preprocessing. Then, each data set is decomposed into a certain number of clusters consisting of similar binary codes, and the binary pattern matching of pairs of malware is limited to each cluster.

Sometimes clustering result in a quite imbalanced decomposition and to obtain more balanced clusters, we introduce the nested clustering method. The main contributions of this research can be summarized as follows:

- Comparing the different clustering algorithms to find the most suitable one for clustering malware binary files.
- 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 nested k-means clustering algorithm with the ordinary binary pattern matching to evaluate the efficiency of nest clustering.

II. GARBAGE DETECTION

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.

A. Binary 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.

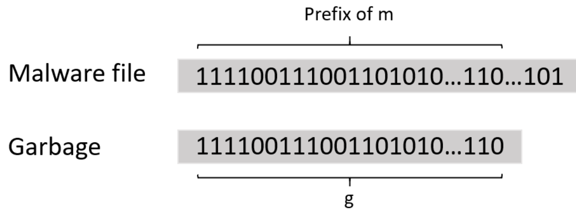

Fig. 1: A garbage comparison with relative malware

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 descending order. Rather than pattern matching the random binary files, making the files in order and comparing them later help in accelerating the matching performance.

The binary files are sorted using the quicksort algorithm, whose worst-time complexity is O(n^2) but average-case complexity is O(n log n). Besides quick sort, we also tried the merge sort algorithm with worst-case complexity of O(n log n). As the innermost loop of the quick sort algorithm is simpler, it can also get about 2 to 3 times faster than the merge sort. Moreover, quicksort 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 quicksort 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 ← QuickSort(S);   #Q[i] ≥ Q[j] ∀i < j
temp ← S[0];
M ← [temp];
G ← [];
for i ← 1 to length(Q) − 1 do
    res ← Matching(Q[i], temp);
    if res is −1 then
        G.append(Q[i]);
        continue;
    else
        M.append(Q[i]);
        temp ← 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 ← bytearray(b.read());
buf_2 ← bytearray(b'.read());
min_length ← minlength(buf_1), length(buf_2);
for i ← 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
```

B. Malware Database

For practical malware analysis, there are some sources that freely provide malware samples. In this research, we collect the data from VirusShare.com website[1], a malware repository collecting, indexing, and freely sharing samples of malware to analysts, researchers, and the information security community. We collect 30 malware data sets from Virusshare, in which some of them contain 131,072 samples and some have 65536 samples in each. To evaluate the scalability for the algorithms, we enlarge the data sets by making the combinations of two to five collected data sets.

[1]https://virusshare.com/

For this research, 55,763 IoT malware samples are supplied by Yokohama National University, which are collected using IoTPOT [2].

C. Garbage Rate

First, we naively tried the binary pattern matching on each collected malware data set to see the exact garbage rate of each set. TABLE I shows the average execution time and average garbage rate percentage of malware data sets with the different number of files inside. The entire result can be checked on Jaist webpages [23].

In VirusShare malware sets, the average percentage of garbage rate of each is around 4%, ranging from 1% 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 significant amount of garbage in it, which is more than 40% of the data set. From TABLE I, we can see that the percentage of garbage composition increases with the number of samples of data set. However, it is tedious and time-consuming to find garbage from large data set only with the binary pattern matching algorithm. To be able to handle the data sets with the enormous number of samples, we intend to speed up the matching process with higher efficiency.

TABLE I: Average number of garbage, garbage rate and execution time of different datasets

No. of File	Avg. Size	Avg. Time	Avg. No. of Garbage	Avg. Garbage Rate
55,763 (IoT)	6.2 GB	1.17 hrs	23,056	41.3%
131,072	40 GB	2.75 hrs	3,670	2.8%
262,144	88 GB	9.50 hrs	6,554	2.5%
393,216	146 GB	15.75 hrs	15,728	4.0%
524,288	191 GB	18.45 hrs	25,165	4.8%
655,360	199 GB	20.00 hrs	35,146	5.4%

III. CLUSTERING AS PREPROCESSING

Instead of comparison on all pairs in a dataset, we try to split the huge data set into groups of similar samples. In this section, we make a comparison among five unsupervised clustering methods: k-means, hierarchical clustering, density-based spatial clustering (DBSCAN), spectral clustering and balanced iterative reducing and clustering using Hierarchies (Birch).

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). 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.

A. 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, having either 65536 or 131072 samples in each. Accuracy is simply computed based on the total amount of garbage found by using binary pattern matching algorithm only.

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

The results of executing clustering algorithms together with binary pattern matching are expressed in TABLE II and TABLE III. All experiments are performed on Ubuntu 18.4 installation powered by an Intel i7-4770 K Core(TM) 3.50 GHz CPU and 32GB of RAM.

B. Result

According to the results from TABLE II, 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 might probably decrease the accuracy of the algorithm.

The performance comparison of the algorithms over bigger dataset is presented in TABLE III. 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%. Although the resulting clusters of every algorithm take similar time for the matching process, the clustering time varies depending on each algorithm. Among them, k-means with linear time complexity takes the shortest time for clustering. 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.

[2]http://www.jaist.ac.jp/~mizuhito/tools.html
[3]http://www.jaist.ac.jp/~s1710443/results.html

TABLE II: Comparison of different clustering algorithms in terms of execution time and accuracy (65,536 files)

Algorithm (+ BPM)	Avg. Clustering Time	Avg. Matching Time	Avg. Accuracy
K-means	2 min	10 min	100%
Birch	2 min	10 min	100%
Hierarchical Clustering	25 min	13 min	100%
DBSCAN	18 min	9 min	80.6%
Spectral Clustering	2 hr	10 min	100%

TABLE III: Comparison of different clustering algorithms in terms of execution time and accuracy (131,072 files)

Algorithm (+ BPM)	Avg. Clustering Time	Avg. Matching Time	Avg. Accuracy
K-means	0.5 hr	1 hr	99.919%
Birch	-	-	-
Hierarchical Clustering	1 hr	1.5 hr	99.886%
DBSCAN	1 hr	1 hr	56.728%
Spectral Clustering	5.5 hr	1.5 hr	100%

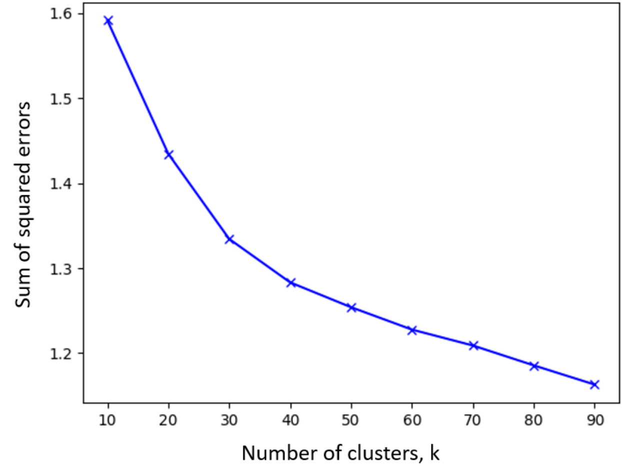

Fig. 2: The elbow method showing the optimal k

IV. PARAMETER SPECIFICATION FOR K-MEANS

The k-means clustering is one of the simplest and frequently used unsupervised learning algorithms, especially in data mining and statistics. In this research, we specify two parameters for the k-means algorithm, which includes:

- Number of clusters, k: being a partitioning algorithm, k-means required the fixed number of clusters to form k groups of data points.
- Input data size: as k-means required the fixed size of input data to compare the similarity (distance difference), we limit the data size of malware binary files

Apart from the above parameters, we make a choice of distance matrix to be used in comparing similarity among the data points. Although there are many other metrics to find the closest distance, we apply commonly used Euclidean distance. As we work on binary data, the Hamming distance would be an alternative choice for finding the distance.

The 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 numbers as the mean values of k clusters, called centroids, and find the nearest data points of the chosen centroids to form k clusters by measuring the distances between each centroid and the data points using the distance matrix. It then 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.

A. *Number of Clusters, k*

The number of clusters should be determined appropriately as they also 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 like 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 [3].

In Fig. 2, 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. Therefore, we chose value 30 for the number of clusters for the clustering process in our further experiments.

B. *Input Data 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 the 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 also makes 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 affect the clustering performance is carried out to select the better one.

In Fig. 3, 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

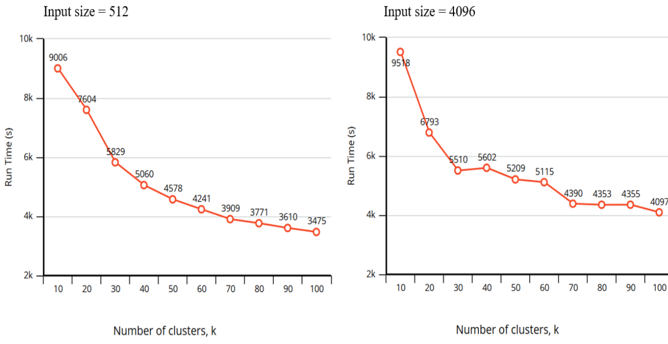

Fig. 3: Comparison of data sizes 512 and 4096 in terms of execution time

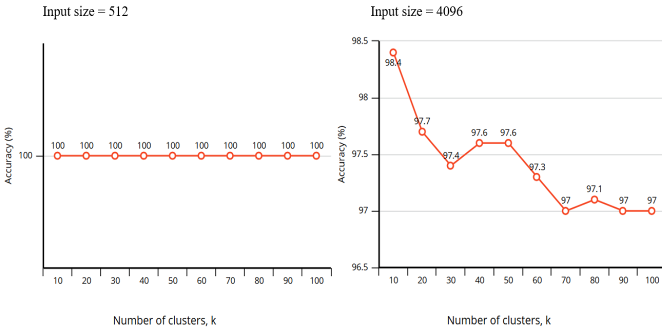

Fig. 4: Comparison of data sizes 512 and 4096 in terms of accuracy

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 of how precisely the algorithm makes 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 Fig. 4, 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.

V. NESTED CLUSTERING

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 numbers of data points in a cluster differ a lot. Large clusters take time in finding garbage. Although k-means works better in clustering than other algorithms, there are some clusters, even ten times larger than the others. To reduce such kind of unbalanced clusters, we develop the nested clustering algorithm.

A. Nested Clustering Algorithm

For nested k-means algorithm, we need to specify two parameters: the number of clusters (k) and the fixed limit 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) If there is a cluster that has more 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. The k-means clustering iterates until the number of files in cluster doesn't exceed the limit f. Note that, there may also be some clusters that cannot be partitioned anymore despite the second trial of k-means. In that case, we avoid the iteration on these clusters.
3) Finally, binary pattern matching is applied on each resulting clusters.
   The clustering and garbage filtering procedures are shown in Algorithm 3.

B. Garbage Detection using Nested Clustering

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 previously observed parameter specifications. TABLE IV present the performance comparison among binary pattern matching and pattern matching after clustering with nested k-means. Apparently, nested k-means works faster and still maintains the high accuracy of found garbage. Clustering cannot be guaranteed that every garbage will be in the same cluster with its related malware file. Therefore, some garbage files may fail to be detected. Experimental results show that the nested k-means algorithm accurately clusters the malware files, which speeds up the pattern matching process two to three times faster while maintaining the high accuracy.

TABLE IV: Performance comparison in terms of time and accuracy

No. of File	BPM	Nested K-means + BPM		
	Avg. Time	Avg. Time	Avg. No. of Missed Garbage	Avg. Acc.
55,763 (IoT)	1.17 hrs	0.75 hr	0	100%
131,072	2.75 hrs	1.50 hrs	2	99.812%
262,144	9.50 hrs	3.75 hrs	3	99.392%
393,216	15.75 hrs	6.75 hrs	6	99.963%
524,288	18.45 hrs	8.30 hrs	10	99.962%
655,360	20.00 hrs	12.45 hrs	14	99.960%

VI. 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. [4]. classifies the freeway operating

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); #C − 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;
```

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. [5] 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 [6]. 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.

VII. CONCLUSION

This research 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 quicksort 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 detects garbage files within a short time with high accuracy. We successfully found out almost every unnecessary garbage from the collected data sets from Virusshare and IoT malware data set.

REFERENCES

[1] 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)

[2] Yin Minn Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow. IoTPOT: A Novel Honeypot for Revealing CurrentIoT Threats, *Journal of Information Processing*, vol.24, no.3, pp. 522-533, (2016)

[3] 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

[4] 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)

[5] 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)

[6] 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)